PREMIERE ISSUE

Volume 1
Number 1

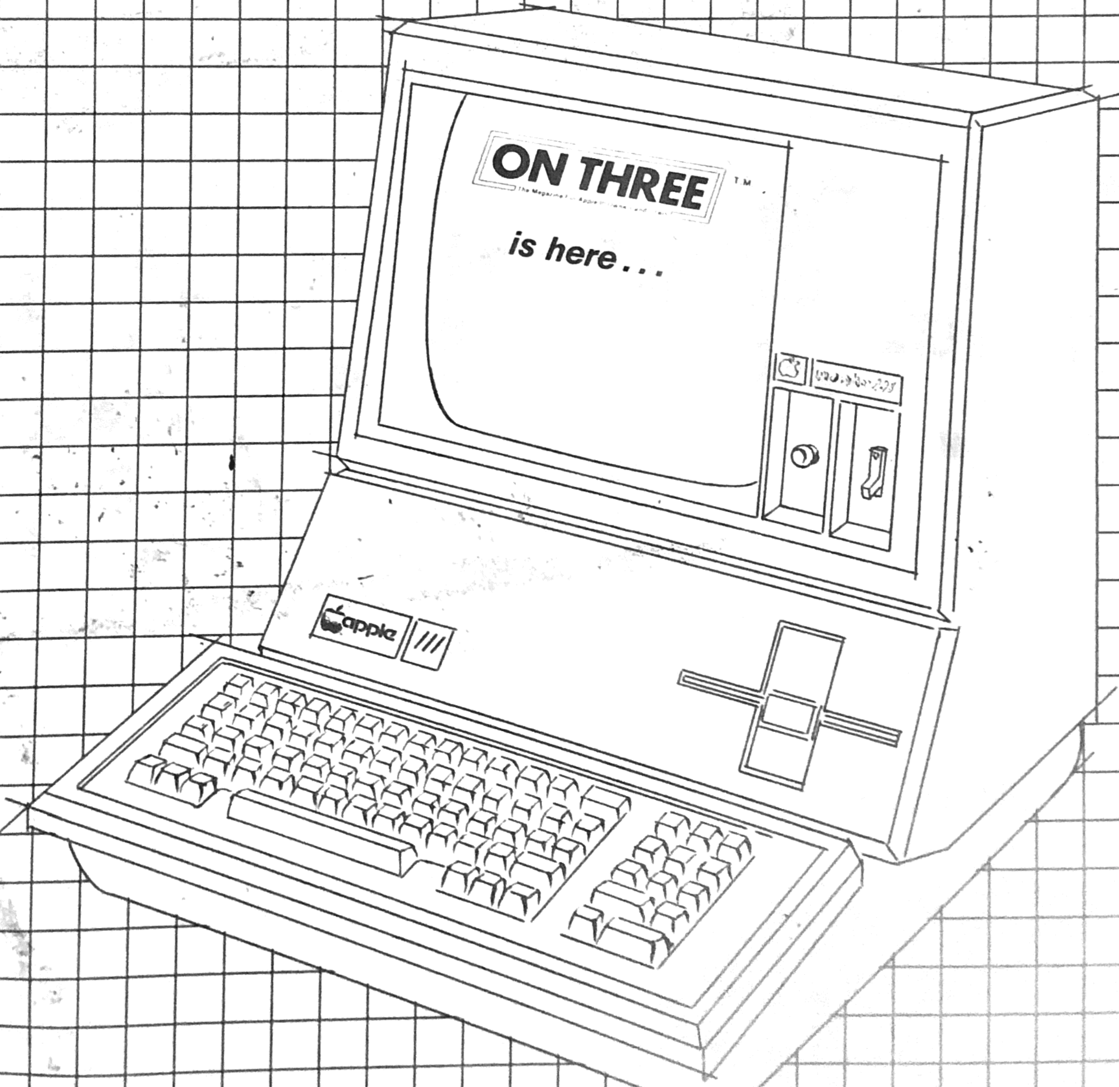
ON THREE™

The Magazine For Apple III Owners and Users

$3.00

January 1983

- Disk Pak 1: Get extra disk space!
- How to tell a ***RETURN*** from an ***ENTER***
- Tutorial, tutorials and more tutorials
- REVIEW: pfs: FILE & REPORT
- Plus much more!



ON THREE T.M.
The Magazine For Apple III Owners and Users

ON THREE
Post Office Box 3825
Ventura, California 93006
(805)644-3514

An open letter to our readers

ON THREE is here! As you page through the magazine you will notice one thing that makes us different from most other magazines. That's right, we do not have any advertising. Why? Well, it's a policy decision that goes something like this.

Everybody who owns a computer has been ripped off at least once by a software or hardware distributer - manufacturer. Either the program or device they bought didn't work at all or it worked only marginally. I know that I can't stop bad programs and peripherals from being sold, but I can prevent their advertising.

Therefore, in the interests of our subscribers, we will devote a special section of the magazine to those products manufacturers have sent us for review.

Once we get a product for review, we will first do a 'mini' review on it to see if the manufacturers claims are correct. If the product turns out to be valid, we will mail the manufacturer an advertising rate sheet and that product may be advertised in the magazine. If we find serious deficiencies we will not include it in manufacturers list. If it is deemed 'Valid' we may or may not do a complete review on it.

Thus, if a product is not in the list, subscribers may assume one of two things. They may assume that either the manufacturer has not yet sent a copy for review, or that they have sent a copy for review and we declined it due to some flaw.

The way I see it, if a manufacturer doesn't want to send in a copy for review, he must have serious questions about his own product, and if the manufacturer has doubts - shouldn't the consumers?

I know that some people may dislike our policy on general principle. It was not an easy decision to make, but I firmly believe that it is in the best interests of our subscribers and therefore, our magazine.

Sincerely,

[illegible]

Bob Consorti
Editor

ON THREE

/// TABLE OF CONTENTS ///

Next Month in *ON THREE*

Listing files in Pascal . . . SOS directory structure revealed (Finally!) . . . Apple Writer III . . . Special *ON* Word Processing Language . . . Pascal Tutorial begins . . . more!

January, 1983
Volume 1 Number 1

Editor/Publisher:
Bob Consorti
Managing Consultant:
Joseph Consorti
Cover Design and Artwork:
William V. Padula
Cranford, New Jersey

Interior Artwork:
Virginia Carol

Printing Services:
Mountain View Press
Oxnard, CA

ON THREE - THE Reference Source for the Apple III is published monthly by *ON THREE,* P.O. Box 3825, Ventura, California 93006.

For a copy of our Author Guidlines, please send a self-addressed stamped envelope (20 cents) to the above address.

Subscription information: U.S. - $30.00 for 12 issues. For first class mail please remit an extra $10.

All foreign subscriptions should include additional postage in the following amounts:
$8 for Canada and Mexico
$12 for Central America - Caribbean
$16 for South America - Europe - Africa
$19 for Asia - Pacific Islands - Australia

Funds may be remitted in either U.S. dollars drawn on a U.S. bank, International Money Order, or by a direct bank draft.

Group rates are as follows:

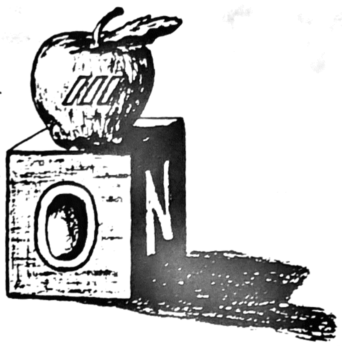
2 - 9 members $28 per subscription
10-49 members $25 per subscription
49 + members $22 per subscription

Return postage must accompany all manuscripts, drawings, and diskettes submitted if they are to be returned, and no responsibility can be assumed for unsolicited materials.

All letters sent to *ON THREE* will be treated as unconditionally assigned for publication and are subject to *ON THREE*'s right to edit and comment editorially.

The Editors Block:

Bob Consorti



Welcome! ON THREE - The Reference source for Apple /// Owners And Users is finally here! I know it has been a long wait for most of you, but now that we are out, we are here to stay.

Who are we, and what are we going to do? If you read my open letter on the inside front cover, you will see that we are a group of concerned Apple /// owners who want to promote what we believe is a very good computer.

Too long have people in the Apple /// community taken a back seat in the industry and watched other computer systems fly by. We are going to do everything in our power to see that people begin to fully utilize their ///'s potential.

As such, we will be a 'Clearing House' for views and ideas about the ///. In the next few months we are going to do reviews, conduct tutorials and bring you programs that will help you get the most out of your Apple ///.

Our reviews will be clear and comprehensive. I believe that before a person spends hundreds of dollars for a program or thousands for a piece of hardware, they should see it in action. Only after making sure that they can use the item, should it be bought.

I do not tolerate software houses who sell products wrapped in sealed plastic, that if you open to look at the manual or test the program, you have to buy it. Policies like these hurt the computing industry and force potential users to either buy the program just to look at it, or do not look - and don't buy!

If dealers and software houses won't let you look before you leap, we will. In a program review, we will show you everything that a particular program can - and can't do. A balanced picture will be presented to give the reader a full view of the items strong and weak points.

Have you ever wanted to learn how to program? Well, our tutorial section may be just what you are looking for. Beginning this month with Basic, and next month with Pascal, you can learn in easy to follow lessons, the language of your choice. Here you will learn step by step the things you need to know to program on the ///.

Each month we will bring you at least one program of "commercial" quality. Thus, your subscription will pay for itself many times over in useful programs that the average person can understand.

Utilities and even some games will be in our repertoire. However, the final word on what we publish will be made by YOU! Please write and tell us what you want to see. Be it a sophisticated sales analysis tool or a color hi-res game, we will try to accommodate you.

Presently we are small, and I can't give you all the things that I wanted to. As the word spreads though, we will grow. Hopefully in a few short months we can present you with the type of magazine that the Apple /// needs - big, bold and innovative. Well, two out of three isn't bad! We will get much bigger as the first subscribers tell others in the Apple /// community about ON THREE, so hang in there!

Getting down to the good stuff!, the premier issue of ON THREE contains a very useful program for those of you who don't have a hard disk. This program will give four extra blocks of space on your data diskettes. Completely compatible with SOS, in the space of a few months I expect it will be copied and sold in other magazines by nameless individuals.

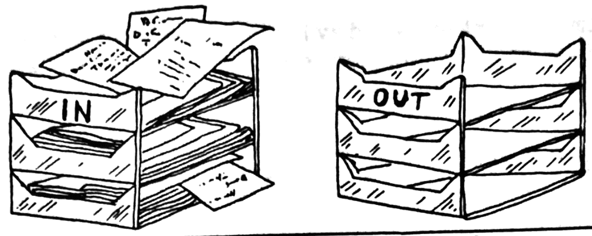
Aren't you lucky? You get the original for the price of a subscription! Also included is an article which shows you how to tell a 'RETURN' from an 'ENTER', read the keypad and determine the state of all the modifier keys. Thus, if you are into programming on the ///, you will now be able to have your programs 'know' when the Alpha Lock, Control, Shift or other special key is being pressed. Included is a very good documentation program, it will help answer any questions you may have.

Next month we will give you a complete review of Apple Writer ///, and explain its unique WPL (Word Processing Language). Also, next time we will present a Pascal program that 'Lists' any directory, in addition to the complete (Finally!) directory structure of SOS disks. Coming up in the months ahead is a program that will restore a deleted file (Lazarus ///!), and even a quick paced game - Little Brick Out ///.

We look forward to providing you with everything you wanted to know about your Apple /// - but didn't know who to ask!

Remember - tell a friend! **///**

Ask THREE:
(Letters to the Editor)



Dear Sirs,

Enclosed is my check for 12 issues of ON THREE. Regarding the Apple /// there are a couple of questions I would like to ask.

1) I have heard that with Pascal a program can be written, compiled and then used in a Basic program through the use of the Invoke and Perform functions. I tried this with a sort routine that compiled and ran in Pascal, but when used in a Basic program an OUT OF MEMORY error occurs. On typing ?Fre there is memory available. Do you have any ideas about this?

2) I am planning to use the Apple /// with some data recording devices through the RS-232 interface but am not sure what's involved with the RS-232 driver. Is another driver routine needed to transfer the data? I'm sure a Basic program is needed to manipulate the data, such as storing on disk, etc. Can you give me any suggestions along this line?

3) I have a Silentype printer but would like to have something a little better. I have seen very little about printers that are compatible with the Apple ///. Also, I would be interested in a spooler to permit use of the computer while the printer was working. Do you have any suggestions?

Thank you very much in advance for any help you can give me. I look forward to receiving ON THREE, we really need something that concentrates on the Apple ///.

Sincerely,

William C. Bell
THAILAND

Dear Mr. Bell,

That's enough questions to fill a magazine, and I think everyone will be interested in the answers, so here goes!

I find the first question and problem very interesting because about a year ago, when I bought the Pascal language system for my ///, I did exactly the same thing!

After much reading I found certain references to 'P-CODE'. What is this mysterious thing?

A few days later I had the answer, it is like the Basic interpreter. It takes an input of certain words and performs certain actions as output. But it does not respond to the same commands as Basic because it does not know Basic. Therefore, a compiled Pascal program cannot be used by a Basic program.

All is not lost however, I said 'compiled' and that is the key. On certain pages of the Program Preparation Tools manual that comes with the Pascal system, there are references to Assembly language. This is the stuff that the computer uses at the lowermost level. The interpreters, like Basic, are written in Assembly language. They take as input a command like 'PRINT 2+3' and convert it into something the computer can understand.

The Basic reference manual tells you how to use external subroutines (assembly language procedures or functions) in your Basic programs. This is the key! Using the Pascal system, you can create 'Assembly language subroutines' that you can use in your Basic program! So a clever programmer can create an assembly language routine that can be used in Basic and Pascal.

In this issue, the article 'Assembling (ON) the ///' gives a good example of an assembly language program that can be used in your Basic programs.

As a final note to the first question, you received an 'Out of Memory' error, because the file you tried to invoke was not in the right format for an invokable Basic module.

In the first few bytes of an invokable module, there is information on how big the file is, where it is to be placed in memory etc. Those bytes told Basic to reserve more space than was available. Basic could not do that so it gave an error message. When you typed ?Fre, Basic checked to see how much memory was left and because the file was never really loaded into memory, it correctly showed there was memory left.

Whew!! One down and two to go!

Question number two is the hardest to answer. Without knowing exactly what type of recorder and the format of its data, I cannot tell you what to do. First of all though, you probably will not need another driver. The RS-232 driver is compatible with just about anything that can transmit on an RS-232-C connector.

Pages 110 and 111 of the Standard Device Drivers manual are examples of reading

Continued on next page

from an external device. Pages 114 through 118 discuss Communications Protocols. To interface a piece of equipment to the ///, you will have to become very familiar with those sections. After an 'INPUT #1; instring$' statement, you will probably have to scan the input string for the data that you want.

The following Basic program will at least show you what you read in, from there you will have to determine by yourself what data to save on disk. Many devices will send special control characters that aren't normally printable, this program will look for these special characters in the input string and print them out in inverse.

```
10 OPEN #1, ".RS232"
20 INPUT #1; instring$
30 FOR position = 1 TO LENGTH (instring$)
40 char% = ASC(MID$ (instring$),
    position, 1)
50  IF char%<32 THEN INVERSE: PRINT char%+
    128;: NORMAL: ELSE PRINT CHR$(char%);
60 PRINT " ";
70 NEXT position
99 END
```

Last one!

Any printer that has an RS-232C interface can be directly connected to the Apple ///. Also, printers that have a parallel interface my be connected to the ///, using the UPI (Universal Parallel Interface) card that Apple sells. Since every printer I know of uses one (or both) of the above interface methods, anything that prints should be compatible with the ///.

Good quality printers that I have seen work with the Apple /// are, Epson's MX-80 and MX-100; Okidata's Microline 82A - 84; NEC 7710,7735; Diablo 630; Starwriter F-10; and the IDS Prism 80, 132.

There are a few methods that allow print spooling. One uses a hardware device. This is a separate piece of equipment that takes the output from your computer as fast as it can send it. As soon as the printer can receive it, the spooler sends out the new material to be printed. Compulink's 'SooperSpooler' is an example. The other method uses software to print out the info. when the computer has time. This slows down the system operation, but most of the time you will never notice it because the computer is so fast. 'Discourse' from Quark uses this method.

I can attest to the workings of Compulink's product because I have it. However I don't have a copy (yet) of Discourse and I can't say anything other than I've heard it works. Hopefully, in the near future we will be able to do a complete review of these products. Until then, let the buyer beware!

That's all! I hope you can understand my ramblings, but if you can't - you can always write me another letter!

Gentlemen,

Enclosed please find my check for $30. This letter is being prepared on my new Apple ///. I have had it about six months and need to learn as much as possible.

I am disappointed that my color monitor is of little use. All the programs that allow me to draw objects in color, become black when using the Apple][emulator. I do not need a game display, but color is a necessity!

Please advise me of what I can do to fix this problem.

Very best wishes,

Robert Scattergood
Florida

Dear Mr. Scattergood,

I know all the frustrations you are going through, I have a color monitor also, and have been frustrated by the lack of color in the Emulation mode. Only the monitor is not at fault, the problem lies with the Apple ///.

If you look on page 144 of the 'Apple /// Owner's Guide', the second paragraph says (rather cryptically) that color hi-res images are not available using the Emulation disk. What this boils down to is that if you want color, you must use the native mode of the ///.

I know this doesn't solve your problem because I know there aren't any programs at the present that allow graphic designing on the ///. However, if you send me a letter stating exactly what you want the program to do, I will publish it at the earliest possible date. Hopefully, someone will respond.

Sincerely,

Bob Consorti

Dear Mr. Consorti,

Thank you very much for the confirmation of my subscription to ON THREE. The name is clever, others with Apple ///'s will spot it right away. (I hope so - Ed.) I am an early retired person, providing

Continued on next page

Horror Stories:

Have you ever been on the wrong end of a deal? Has a dealer or manufacturer ever sold you a bad product and refused to give you satisfaction? This is where you can get back at them! Tell us all how you were given the run around, and maybe then it will be resolved.

This months entry comes from me! Yes, I got taken to the cleaners once. It involves an Amdek Color-II monitor, two dozen phone calls and twelve months of waiting.

Early January of last year, I bought the monitor from a reputable dealer in L.A. after they were told by the company that it directly interfaces with the Apple ///. I think you can guess the rest. That's right, not only did it not work with the ///, the company kept telling me it would be just a few weeks before the appropriate interface box would be available.

All of my calls to the company left me with the same impression - they in effect were saying, "Hey that's your problem now". They would not supply me with schematics so I could build an interface. For six months I had a thousand dollar monitor sitting in my closet. SIX MONTHS!!

Sometime in June my dealer informed me they had received the interface and I could pick it up. Do you think we are done? No way! I not only had to pay forty dollars to get the darned thing, but it did not really work. As soon as I got it home I tried it out. With a colored background I counted the colors - 1, 2, 3...8! Only eight colors on my Apple /// with this monitor.

More calls revealed that they were coming out with a Color-IIA, which would allow Apple /// users to get the full sixteen colors that the /// can display. Could I trade in my present monitor and get the new one? Of course not! They said they were going to come out with an update kit for the older Color-II's. In October it finally arrived. It didn't work. I tried and my dealer tried to make it work. It wouldn't.

Where are we now? Well, I have got a monitor that I can't really use, and some very negative feelings about Amdek. Therefore, until they choose to make my monitor work or give me back my money, I would suggest that our readers be very careful when buying anything from them. If they ever do rectify the situation we will publish any response that they choose to give. ///

Continued from page 4

others with a place to come in the winter that does not require a snowshovel. In this role I am often needed to perform as an architect,interior decorator and all around handy-man for those whose husbands have passed on.

The program that I would like, must enable me to draw a room outline, complete with windows, door locations, etc. I would also like to draw blocks of different shapes and colors on the screen to represent the pieces of furniture. These blocks or colored outlines should have the ability to be moved under cursor or paddle control.

I would like to know in advance of any purchase, the needed placement and whether it would go through the window or the door. The program should be able to tell me if a given piece of furniture will fit along the wall space between windows or a door.

Just think of all the aching backs that would be assisted when the 'little woman' says, "No..this is no good,..Put the furniture back where it was" - Whew!

It should also have the ability to [S]ave the furniture information with the color coding on disk for later use. A print out of the screen would be nice too. I would also like to be able to match names and apartment numbers with the data of the room designs on the disk.

I challenge the program advertisers to have a (C) before each program that allows an Apple /// with a color monitor to display in color (Like TV Guide used to do!).

Thank you for an Apple /// publication,

Robert Scattergood

How about it, all you programmers out there? I know that a 3-D graphics package will be released soon, but I don't think it will do anything like what Mr. Scattergood wants. Please write to ON THREE if you have any information.

We are here to help clear up any problems you may be having, so if you are a customer (or dealer) and have a problem, write us! We may not be able to help, but we will certainly do our best! ///

Disk Pak1: Extra Disk Space Plus!

by Bob Consorti

Does this situation sound familiar? Typing data into the computer, you try to add one more record into your database. If you are lucky and are using a good program, you will get some kind of an error message indicating that there is no more room left on the disk. If you are not so lucky, your program bombs and leaves your disk file in an unrecoverable mess.

With the price of hard disk and high density floppies falling to within reach of more and more consumers, this scenario will occur less frequently. However, until the price of those 20 megabyte drives drop down to under a thousand dollars, a very large percentage of people will be using the standard old 140K diskettes.

I would like to be able to write you an article of how to turn your built in drive into a multi megabyte Winchester, but I haven't figured out how to do that yet. What I can do is tell you how you can get four extra blocks of storage space on a standard floppy!

Using the program presented on the next few pages, you will be able to make data diskettes that can hold 277 blocks of info. Yes, a normal diskette can only store 273 blocks. While this is a less than two per cent improvement, it's a very important two per cent.

. . . these new disks are completely compatable with standard SOS.

The best part about it is that these new disks are completely compatible with standard SOS. This means that you can put a bigger Visicalc template on a regular floppy. You will be able to put larger textfiles, datafiles - any kind of file, on a standard floppy.

Wait a minute!. How does this guy do it? With mirrors! No, I was just kidding. There is a drawback which I will explain in a minute. You may have noticed that I used the term 'data diskettes'. The problem is, on a normal diskette a certain amount of space is reserved for the directory. It normally takes up four blocks of space and has room to store up to 51 file names.

If you have just one very big file on your data diskettes, you can see that the space for all those names on the directory is a waste. What I have done is reclaim three of the blocks used for the directory. This leaves us with a disk that can hold more info., but the directory now has room for only 12 file names. Thus, if you have a whole bunch of small files, it would be best to use a normal diskette.

If you'll remember I said we can gain four extra blocks, so where are we going to get the other block? Well, I think we all know what the term 'Booting a disk' means. Whenever a disk is 'Booted', the computer first reads into memory the information located on block zero of that disk. This information is what they call the 'Second Stage Boot' program. Once it is in memory, control is passed to it and it performs the rest of the 'boot' by reading in the files that are needed to operate any language on the /// - SOS.KERNEL, DRIVER and INTERP.

On the Apple][(Don't worry we will get to answering the question!), the Pascal 'Second Stage Boot' program takes up two blocks of disk space. However, on the ///, because the machine is much more sophisticated, this program takes up only 1 block. But somebody in the back room at Apple decided to leave space on the disk for a 'Second Stage Boot' program that was up to 2 blocks long.

What does all this mean? Well, on every standard format diskette that the /// uses, there is one block of space that is never used. This is the other block that we can gain. I know that various people at Apple will be reading this article, so I implore them to fix this little bug. Even with just one extra block of disk space, people will be happier!

The good stuff!

A while back I decided to do more than just give you a program that just modifies a disk so you can get extra space. The program presented on the next few pages will also add and/or change the diskettes volume number. This number is not normally shown using Pascal, but Basic - Apple Writer /// and many other languages can show you it.

Also, when a diskette is formatted for use, the date and time that it was created

Continued on next page

are supposed to be marked on it. Again, this info. can be seen using the various languages of the ///. I said 'supposed' because it isn't! Here is a small bug that gets some people (including me) angry. Whenever I catalog a disk I like to see when I formatted it. Diskettes, being as fragile as they are, have been known to deteriorate over time. If I know that a particular diskette has been in use for some time, I like to make a copy and then re-format it to make sure it's sound.

This program also includes the capability of changing the diskettes date and time of creation. Since no program is ever 'finished' and this one is no exception, I know there are improvements that can and will be made in the future. I've even done some things inefficiently on purpose! As is stated in the program listing, we will encourage any and all improvements and will publish them as we receive them.

Turning now to the program, the first seven procedures and functions are self explanatory. They are simple housekeeping operations. The meat of the program occurs in three procedures. 'Four_more_blocks', which performs the above described modifications to get four extra blocks of space on a diskette. 'New_volume', which changes a diskettes volume number, and 'New_times' which changes the date and time info.

The procedure 'Four_more_blocks' is really very simple. It checks to see if the disk has already been updated or if there are files on it. If either of these are true, no update occurs and the user is notified of that. When a standard disk with no files on it is used, the program updates the disk with no problems.

The procedure 'New_volume' is also straight - forward. It first displays the current volume number, and then asks the user if he or she wants to change it. If the answer is yes, the program asks for a new volume number and updates the disk. Otherwise, the procedure is exited without an update. Because only one byte is used to store the volume number on the diskette, it must be in the range of 0 to 255.

The last procedure we will discuss is the most complicated. 'New_times' will update the date and time of creation that is stored on each diskette. First it gets the info. from the disk and then displays it for the user to see. If the user wants to change any or all of this info., they are then prompted to enter the new dates and times. Only if all of the input is within bounds (the month record must be from 0-12, etc.) is the diskette updated.

You may have noticed that I used the function 'Str_to_int' extensively throughout the program. This is a handy little routine that takes as input a string and returns the integer representation of that string. It is used to 'idiot proof' the input so that accidentally typing a Q when you meant a 1 will not bomb the program.

To use this program, type it in and compile it. Since the program will only give those four extra blocks to an empty (formatted, but no files) diskette, format some blank diskettes using the System Utilities Disk. Once executed, you can modify as many diskettes as you wish. Remember, you can change the time, date and volume number on any diskette, not just the blank ones you just formatted.

These modified disks can be used by virtually any program. Used with PFS:FILE, the disks can now store about 64 more forms (records) per disk. Bigger forecasts for you Visicalc users, extra room on each disk for those of you doing word processing, and more space for everyone!

These modified disks can be used by virtually any program.

Watch out, though. I have not checked it yet, but there is a routine that transfers Mail List Manager files to the format that Apple Writer /// uses, and I think it uses absolute positions on the disk when doing the transfer. Since this program changes the structure of the information on the storage diskette, you may want to try it out on a backup copy first. I'd hate to have you lose some important data.

When using the Pascal filer or System Utilities Program to list the directory of a disk that you have modified, don't worry when you get the error message indicating that the directory structure is damaged. Nothing is wrong, and since we did change the disk - I would be upset if it did not give that message.

Have fun with your new disks. In the next issue I will give you a Pascal program that allows you to catalog a disk, along with the complete directory structure of SOS disks, which makes it easy! Remember, try this program on a blank disk first. You would be amazed at what one misplaced word, number or character can do to your disks. Therefore, please follow the first rule of computing - 'Back-up Thine Stuff'! **///**

Continued on next page

Extra Disk Space Plus!: Program Listing

```
PROGRAM Four_extra_blocks_and_more;

{ ******************************************************************************** }
{ *                                                  ---------------------------   * }
{ *   Disk Pak1: Extra disk space & more!            |  Copyright 1982, 1983 by |  * }
{ *   -----------------------------------            |      O N   T H R E E     |  * }
{ *   by Bob Consorti                                |      January, 1983       |  * }
{ *                                                  ---------------------------   * }
{ *   This program will take a diskette that was formatted by the System           * }
{ *   Utilities Disk and modify it so that four extra blocks of space are          * }
{ *   available for use.  It will also change the diskette volume number           * }
{ *   and time and date of creation, which are not set when a disk is first        * }
{ *   formatted.                                                                   * }
{ *                                                                                * }
{ *   Improvements and modifications are encouraged and will be published          * }
{ *   in a future issue.                                                           * }
{ *                                                                                * }
{ ******************************************************************************** }

CONST Device_num = 4;         { The Pascal unit # of the built in drive }
      Block_length = 512;     { The size of a block of data on the disk }
      Block_mode = 12;        { Disables all options of UNIT read/write }
      Bell = 7;               { Causes a beep on the internal speaker    }
      Top_viewport = 2;       { Sets the top of the currently defined viewport }
      Move_cursor_up = 11;    { Just like it says }
      Clear_viewport = 28;    { Homes the cursor and clears the viewport }
      Clear_to_end_of_viewport = 29; { Again, just like it says }

TYPE Counter = INTEGER;

VAR Block_num: INTEGER;      { Set to the block # just read or written }
    Block_buf: PACKED ARRAY [1..Block_length] OF 0..255;
                              { Creates space for the block of data }
    Next, Done: BOOLEAN;      { Used to control the program flow }
    Err_message: STRING;      { Used whenever an error occurs }


PROCEDURE Set_titles;         { Sets the main page heading for the entire program }
VAR i: Counter;

BEGIN
  WRITE (CHR (Clear_viewport));
  GOTOXY (0, 0);
  WRITE ('Disk Utility Pak1');
  GOTOXY (60, 0);
  WRITELN ('Copyright 1982, 1983');
  WRITE ('by Robert Consorti');
  GOTOXY (64, 2);
  WRITELN ('by ON THREE');
  FOR i := 1 TO 10 DO
    WRITE ('--------');
  WRITE (CHR (Top_viewport))
END; { of PROCEDURE Set_titles }
```

Program listing continued on next page.

```
PROCEDURE Option (Title: STRING);  { Sets the menu display title }
VAR i: Counter;

BEGIN
  WRITELN (CHR (Clear_viewport));
  WRITELN (Title);
  FOR i := 1 TO LENGTH (Title) DO
    WRITE ('-');
  WRITELN;
  WRITELN
END; { of PROCEDURE Option }


PROCEDURE Prompt_start;  { Prompts the user to put in the disk to be updated }
VAR i: Counter;
    ch: CHAR;

BEGIN
  WRITELN ('Insert the disk you want to update into the built in drive.');
  WRITE ('Press <RETURN> when ready');
  READLN (ch);
  FOR i := 1 TO 3 DO
    WRITE (CHR (Move_cursor_up));
  WRITE (CHR (Clear_to_end_of_viewport))
END; { of PROCEDURE Prompt_start }


PROCEDURE Prompt_done;  { Gives the message of whether or not the update }
BEGIN                   { was successful. }
  WRITELN;
  IF Done THEN
    WRITELN ('Update procedure completed successfully')
  ELSE
    WRITELN ('No update this time')
END; { of PROCEDURE Prompt_done }


PROCEDURE Error;  { General purpose routine that prints out an error message }
BEGIN
  WRITELN (CHR (Bell));
  WRITELN (Err_message)
END; { of PROCEDURE Error }


FUNCTION Another_one: BOOLEAN;  { Asks the user if he or she wants to }
VAR ch: CHAR;                   { update another disk }

BEGIN
  WRITELN (CHR (Bell));
  WRITE ('Do you want to update another disk (Y or N)? ');
  READ  (ch);
  WRITELN;
  IF (CH IN ['Y', 'y']) THEN
    Another_one := TRUE
  ELSE
    Another_one := FALSE
END; { of FUNCTION Another_one }
```

Program listing continued on next page.

```
PROCEDURE Insert_system_disk;   { When we are done, have the user put in the }
VAR ch: CHAR;                   { boot disk.  Otherwise, the system will crash }

BEGIN
  WRITELN (CHR (Clear_viewport));
  WRITELN ('Insert SYSTEM disk in the built in drive');
  WRITE ('and press <RETURN> when ready');
  READLN (ch)
END; { of PROCEDURE Insert_system_disk }


FUNCTION Directory_check: BOOLEAN;  { Checks to see if the disk has already }
CONST First_link = 3;               { been updated or not }
      No_files = 0;

VAR Dir_link1, Num_files: INTEGER;

BEGIN
  Block_num := 2;
  UNITREAD (Device_num, Block_buf, Block_length, Block_num, Block_mode);
  Dir_link1 := Block_buf [3];
  Num_files := Block_buf [38];
  IF ((Dir_link1 <> First_link) OR (Num_files <> No_files)) THEN
    BEGIN
      Directory_check := FALSE;
      IF (Dir_link1 <> First_link) THEN
        Err_message := 'Non-standard disk, it''s probally already updated.'
      ELSE
        Err_message := 'This disk has files on it!!, can not update.'
    END
  ELSE
    Directory_check := TRUE
END; { of FUNCTION Directory_check }


FUNCTION Bit_map_check: BOOLEAN;  { Another check to see if the disk has }
CONST All_blocks_empty = 0;       { already been updated }
      One_block_full   = 1;

VAR Bit_map1: INTEGER;

BEGIN
  Block_num := 6;
  UNITREAD (Device_num, Block_buf, Block_length, Block_num, Block_mode);
  Bit_map1 := Block_buf [1];
  IF ((Bit_map1 = All_blocks_empty) OR (Bit_map1 = One_block_full)) THEN
    Bit_map_check := TRUE
  ELSE
    BEGIN
      Bit_map_check := FALSE;
      Err_message := 'Non-standard disk, it''s probally already updated.'
    END
END; { of FUNCTION Bit_map_check }

PROCEDURE Update_directory;  { The first half of the update procedure }
BEGIN
  Block_num := 2;
```

Program listing continued on next page.

```
  UNITREAD (Device_num, Block_buf, Block_length, Block_num, Block_mode);
  Block_buf [3] := 0;
  UNITWRITE (Device_num, Block_buf, Block_length, Block_num, Block_mode);
END; { of PROCEDURE Update_directory }


PROCEDURE Update_bit_map;  { The second half of the update procedure }
CONST All_blocks_empty = 0;
      Four_blocks_empty = 92;

VAR Bit_map1: INTEGER;

BEGIN
  Block_num := 6;
  UNITREAD (Device_num, Block_buf, Block_length, Block_num, Block_mode);
  Bit_map1 := Block_buf [1];
  IF (Bit_map1 = All_blocks_empty) THEN
    Block_buf [1] := Four_blocks_empty
  ELSE
    Block_buf [1] := Four_blocks_empty + 1;
  UNITWRITE (Device_num, Block_buf, Block_length, Block_num, Block_mode)
END; { of PROCEDURE Update_bit_map }


PROCEDURE Four_more_blocks;  { Gives the extra disk space, if possible }
CONST ok = TRUE;

BEGIN
  Option ('Extra Disk Space');
  Prompt_start;
  IF ((Directory_check = ok) AND (Bit_map_check = ok)) THEN
    BEGIN
      Update_directory;
      Update_bit_map;
      Done := TRUE;
    END
  ELSE
    BEGIN
      Error;
      Done := FALSE
    END { of ELSE BEGIN }
END; { of PROCEDURE Four_more_blocks }


FUNCTION Str_to_int (num_str: STRING): INTEGER;  { Converts the input string }
CONST Plus  = 1;                                 { into an integer.  Used to }
      Minus = -1;                                { 'idiot-proof' the input.  }

VAR Sign, Power, Place_num, Temp_int: INTEGER;
    i, Pos_cnt: counter;

BEGIN
  Sign := Plus;
  IF Num_str [1] IN ['-', '+'] THEN
    BEGIN
      CASE Num_str [1] OF
        '+': Sign := Plus;
```

Program listing continued on next page.

```
         '-': Sign := Minus
        END; { of CASE ... statement }
        DELETE (Num_str, 1, 1)
      END;
    Temp_int:= 0; Pos_cnt := 0;
    FOR i := LENGTH (Num_str) DOWNTO 1 DO
      BEGIN
        Place_num := ORD (Num_str [i]) - ORD ('0');
        Pos_cnt := Pos_cnt + 1;
        Temp_int := Temp_int + TRUNC (PWROFTEN (Pos_cnt - 1)) * Place_num
      END;
    Str_to_int := Sign * Temp_int
  END; { of FUNCTION Str_to_int  }


  PROCEDURE New_volume;  { Changes the disk volume number }
  VAR Volume_num: INTEGER;
      ch: CHAR;


      PROCEDURE Do_volume;  { Performs the actual disk update }
      VAR Volume_str: STRING;

      BEGIN
        Done := TRUE;
        WRITELN;
        WRITE  ('Enter the new volume # (0-255) --> ');
        READLN (Volume_str);
        IF (Volume_str = '') THEN
          Done := FALSE
        ELSE
          BEGIN
            Volume_num := Str_to_int (Volume_str);
            Block_buf [33] := Volume_num MOD 255;
            UNITWRITE (Device_num, Block_buf, Block_length, Block_num, Block_mode)
          END { of ELSE BEGIN }
      END; { of PROCEDURE Do_volume }



  BEGIN { Main of New_volume }
    Option ('Get new Volume number');
    Prompt_start;
    Block_num := 2;
    UNITREAD (Device_num, Block_buf, Block_length, Block_num, Block_mode);
    Volume_num := Block_buf [33];
    WRITELN ('Current Volume # of the disk in drive 1 is #', Volume_num);
    WRITELN;
    WRITE   ('Do you wish to change it (Y or N)? ');
    READ  (ch);
    WRITELN;
    IF (ch IN ['Y', 'y']) THEN
      Do_volume
    ELSE
      Done := FALSE
  END; { of PROCEDURE New_volume }
```

Program listing continued on next page.

```
PROCEDURE New_times;  { Changes the disk time & date of creation }
VAR Date_1, Date_2, Minute, Hour, Month, Day, Year: INTEGER;
    Str_mnth, Str_day, Str_yr, Str_hr, Str_min: STRING;
    ch: CHAR;
    ok: BOOLEAN;


    PROCEDURE Get_times;         { Gets the time information from the }
    BEGIN                        { input block buffer }
      Date_1:= Block_buf [29];
      Date_2:= Block_buf [30];
      Minute:= Block_buf [31];
      Hour:= Block_buf [32];
      Month:= TRUNC (Date_1 / 32);
      Day:= TRUNC ((Date_1 / 32 - Month) * 32 + 0.5);
      Year:= TRUNC (Date_2 / 2 + 0.5);
      IF ODD (Date_2) THEN
        BEGIN
          Month:= Month + 8;
          Year:= Year - 1
        END { of IF ... THEN loop }
    END; { of PROCEDURE Get_times }


    PROCEDURE Fix_dates;  { Adds any needed leading zeroes }

        PROCEDURE Change_str (Num: INTEGER; VAR Str_type: STRING);
        BEGIN
          STR (Num, Str_type);
          IF (LENGTH (Str_type) = 1) THEN
            INSERT ('0', Str_type, 1)
        END; { of PROCEDURE Change_str }

    BEGIN { Main of fix_dates }
      Change_str (Month, Str_mnth);
      Change_str (Day, Str_day);
      Change_str (Year, Str_yr);
      Change_str (Hour, Str_hr);
      Change_str (Minute, Str_min)
    END; { of PROCEDURE fix_dates }


    PROCEDURE Change;  { Performs the time & date changes }
    VAR i: Counter;
        Bad: BOOLEAN;

        PROCEDURE Try (Str: STRING; VAR Spec_type: INTEGER; Low, High: INTEGER);
        VAR In_str: STRING;

            FUNCTION Check (Num_type: STRING; C_low, C_high: INTEGER): BOOLEAN;
            BEGIN
              IF Str_to_int (Num_type) IN [C_low..C_high] THEN
                BEGIN
                  ok := TRUE;
                  Check := TRUE
                END
```

Program listing continued on next page.

```
          ELSE
            BEGIN
              Bad := TRUE;
              Check := FALSE
            END { of IF ... THEN ELSE clause }
        END; { of FUNCTION Check }

    BEGIN { Main of Try }
      WRITE ('Enter new', Str:7, ' of creation (', Low, '..', High, ') - ');
      READLN (In_str);
      IF (In_str = '') THEN
        ok := TRUE
      ELSE
        IF Check (In_str, Low, High) THEN
          Spec_type := Str_to_int (In_str)
    END; { of PROCEDURE Try }


BEGIN { Main of Change }
  WRITELN;
  WRITELN ('Press RETURN to accept the default shown above');
  WRITELN;
  Bad := FALSE;
  Try ('month', Month, 0, 12);      { Makes sure 'Month'  is from 0 - 12 }
  Try ('day', Day, 0, 31);          { Makes sure 'Day'    is from 0 - 31 }
  Try ('year', Year, 0, 99);        { Makes sure 'Year'   is from 0 - 99 }
  Try ('hour', Hour, 0, 23);        { Makes sure 'Hour'   is from 0 - 23 }
  Try ('minute', Minute, 0, 59);    { Makes sure 'Minute' is from 0 - 59 }
  IF Bad THEN
    BEGIN
      WRITELN;
      WRITELN ('Bad input, please re-enter...');
      FOR i := 0 to 2000 DO
        ok := FALSE;                { Waste a little time before going back }
      GOTOXY (0,8);
      WRITE (CHR (Clear_to_end_of_viewport))
    END { of IF ... ELSE BEGIN }
END; { of PROCEDURE Change }


PROCEDURE Do_times;  { Performs the actual time changes }
BEGIN
  Done := TRUE;
  REPEAT
    Change
  UNTIL (ok);
  Block_buf [29] := (Day + Month * 32) MOD 256;
  Block_buf [30] := Year * 2;
  IF (Month >= 8) THEN
    Block_buf [30] := Block_buf [30] + 1;
  Block_buf [31]:= Minute;
  Block_buf [32]:= Hour;
  UNITWRITE (Device_num, Block_buf, Block_length, Block_num, Block_mode)
END; { of PROCEDURE Do_times }
```

Program listing continued on next page.

```
BEGIN { Main of New_times }
  Option ('Change Time & Date');
  Prompt_start;
  Block_num := 2;
  UNITREAD (Device_num, Block_buf, Block_length, Block_num, Block_mode);
  Get_times;
  WRITELN ('Date and time of creation:');
  WRITELN ('Month/Day/Year  Hour:Minute');
  Fix_dates;
  WRITELN (Str_mnth: 5, '/', Str_day, '/', Str_yr, Str_hr: 9, ':', Str_min);
  WRITELN;
  WRITE ('Change (Y or N)? ');
  READ (ch);
  WRITELN;
  ok := TRUE;
  Done := FALSE;
  IF ch IN ['Y', 'y'] THEN
    Do_times
END; { of PROCEDURE New_times }


PROCEDURE Do_options;  { Show the main menu and perform the requests }
VAR Answer: CHAR;

BEGIN
  Option ('Options menu');
  WRITELN ('1 - Get extra disk space');
  WRITELN ('2 - Add or change the disk volume number');
  WRITELN ('3 - Change the disk date & time of creation');
  WRITELN;
  WRITE   ('Enter 1, 2 or 3 --> ');
  READ (Answer);
  WRITELN;
  CASE Answer OF
    '1': Four_more_blocks;
    '2': New_volume;
    '3': New_times
    OTHERWISE
      BEGIN
        WRITELN;
        WRITELN (CHR (Bell), 'Invalid answer');
        Done := FALSE
      END { of OTHERWISE clause }
  END { of CASE ... }
END; { of PROCEDURE Do_options }


BEGIN { Main program }
  Set_titles;
  REPEAT
    Do_options;
    Prompt_done;
    Next := Another_one;
  UNTIL (Next = FALSE);
  Insert_system_disk
END. { of PROGRAM Four_extra_blocks_and_more }
```

Assembling (ON) the ///

by Martin Nichols

You're sitting alone at your computer, working on a program that is a combination garage door opener and sales analysis tool. Staring off into space you decide to add all sorts of special function keys for the user to control the program with. "Wouldn't it be nice to have the program be able to distinguish between the RETURN and ENTER keys", you mutter as you begin typing.

Shouldn't be too hard - right? So off you go trying to figure out how to do it. A few hours later you decide it's too much work. So what if the user has to type 'CONTROL CLOSED APPLE TAB' while standing on one foot and baying at the moon to make the program work!

Well, all you 'Closet' programmers can come on out and take a look here! With this assembly language program, you will be able to tell a 'RETURN' from an 'ENTER' and do all sorts of other neat things.

These routines are based on the fact that each time you press a key on the ///, two bytes of keyboard data are generated. They use this fact to give the user easy to use invokable (or linkable - for you Pascal buffs) functions that make something complex, very easy.

You will now be able to tell the difference between a key pressed on the numeric keypad and a key on the main keyboard. One of the functions returns the second byte of data, and another one will tell you if the ENTER key was pressed.

Using the second byte of data you will be able to monitor the status of all the 'Special' keys on the keyboard. You will be able to tell when the Alpha-Lock, Shift, Control, Open Apple, and Closed Apple keys are being pressed.

This will give your program access to hundreds of combinations of 'Special Function Keys'. For example, have the user type 'Closed Apple C' to catalog a disk, 'Closed Apple D' to delete a file. These, and many others are possible with this assembly language routine.

You can now use special combinations of keys to perform certain functions, just like those expensive programs do!

To be able to use it in your programs, use the Pascal editor to enter the assembly language programs in listing #2. Assemble it and name it 'Key.Things' or 'Keypress' - something like that. Next, boot a Basic disk and type in the 'Invokable Module' documentation (program listing #3). This program will give you a lot more information on how to use these routines.

Below, you will see program listing #1. This is a short example of how to use the second byte of data to determine the state of all the special keys. Type it in and run it. You can press any of those keys and the program will know it!

Next month I will present a few other assembly language functions to make this even easier, along with a couple of extra goodies. But, if you have any special tricks that you would like to see your Apple /// do, write me in care of ON THREE. I'll see what I can do. ///

Key–Things: Program Listing#1

```
0   REM ************************************************************
1   REM *                                 ---------------------- *
2   REM *  Byte-two Test                  | Copyright 1982, 1983 | *
3   REM *  -------------                  |     O N   T H R E E  | *
4   REM *  by Martin Nichols              ---------------------- *
5   REM *  This program will read the state of all of the        *
6   REM *  'Special' keys.  Make sure the invokable module       *
7   REM *  is available and RUN it.  Try pressing the Apple      *
8   REM *  keys, Alpha Lock, etc.  The program knows it!         *
9   REM ************************************************************
10   INVOKE "KEYPRESS.INV":GOTO 50
20   VPOS=num%:NORMAL:PRINT key$(num%):num%=num%+1:RETURN
30   VPOS=num%:INVERSE:PRINT key$(num%):num%=num%+1:RETURN
50   key$(1)="Special key (Keypad, Arrows, Space, Escape
  and Tab keys)"
60   key$(2)="Keyboard is on":key$(3)="Closed Apple"
70   key$(4)="Open Apple":key$(5)="Alpha lock"
80   key$(6)="Control":key$(7)="Shift":key$(8)="Any key"
90   TEXT:HOME:ON KBD GOSUB 200
100   VPOS=10:PRINT USING"56C";"PRESS 'CONTROL Q' TO EXIT"
110   b2%= EXFN%.bytetwo:num%=1
120   IF b2%>127 THEN GOSUB 30:b2%=b2%-128:ELSE GOSUB 20
130   IF b2%>63 THEN GOSUB 30:b2%=b2%-64:ELSE GOSUB 20
140   IF b2%>31 THEN GOSUB 20:b2%=b2%-32:ELSE GOSUB 30
150   IF b2%>15 THEN GOSUB 20:b2%=b2%-16:ELSE GOSUB 30
160   IF b2%>7 THEN GOSUB 20:b2%=b2%-8:ELSE GOSUB 30
170   IF b2%>3 THEN GOSUB 20:b2%=b2%-4:ELSE GOSUB 30
180   IF b2%>1 THEN GOSUB 30:b2%=b2%-2:ELSE GOSUB 20
190   IF b2% THEN GOSUB 30:ELSE GOSUB 20
199   GOTO 110
200   IF KBD=17 THEN NORMAL:HOME:END:ELSE ON KBD GOSUB 200
210   RETURN
```

Continued on next page

Key-Things: Program Listing#2

```
; **********************************************************************
; *                                              ----------------------------  *
; *  Key-things:  How to have fun with the      ! Copyright 1982, 1983 by !  *
; *               second byte of key info.      !      O N   T H R E E     !  *
; *  -----------------------------------        !     January, 1983       !  *
; *  by Martin Nichols                           ----------------------------  *
; *                                                                            *
; *  These assembly language routines will enable the Basic or Pascal          *
; *  user to easily determine  1: If a key on the numeric keypad has been     *
; *  pressed, 2: The second byte of keyboard data, and 3: If the 'ENTER'       *
; *  key was just pressed.  Useful for programs that distinguish between       *
; *  the 'RETURN' and the 'ENTER' key.                                         *
; *                                                                            *
; *  To use in your Basic programs, assemble this routine using the Pascal     *
; *  assembler, and then invoke it as you would any other invokable module.   *
; *                                                                            *
; *  For use in Pascal, declare each of these routines, EXTERNAL FUNCTIONS     *
; *  and use the linker to link them into your Pascal host program.            *
; *                                                                            *
; **********************************************************************

        .MACRO  Pop                 ;Pull a word from the stack
        PLA
        STA     %1
        PLA
        STA     %1+1
        .ENDM

        .MACRO  Push                ;Push it back on
        LDA     %1+1
        PHA
        LDA     %1
        PHA
        .ENDM

        .MACRO  Open
        POP     Return
        PLA                         ;Discard 4 bytes stack bias
        PLA                         ;(for .FUNC only)
        PLA
        PLA
        LDA     #00                 ;Zero the address 'Result' cause
        STA     Result              ;you never know what it may contain.
        STA     Result+1
        PHP                         ;save status, then disable interrupts
        SEI
        LDA     Envrmt              ;save environment
        STA     Env
        LDA     #73                 ;Use a new environment register
        STA     Envrmt              ;Do this to get at the 'C000' I/O space.
        .ENDM

        .MACRO  Close
        LDA     Env                 ;restore environment
        STA     Envrmt
```

Program listing continued on next page.

```
        PLP                         ;restore status (including interrupts)
        Push    Result              ;Give an answer
        Push    Return              ;Come back from non SOS-land
        .ENDM

Return  .EQU    0
Result  .EQU    2
Env     .EQU    4
Kbd     .EQU    0C000
Kbdflag .EQU    0C008
Kbdstb  .EQU    0C010
Envrmt  .EQU    0FFDF

;                   - Function Keypad -
;
;If a key ('0' - '9', '.', '-' or 'ENTER') on the numeric keypad
;is pressed, the BASIC statement 'int% = EXFN%.Keypad' will return
;the Ascii value of the key pressed in the variable 'int%'. If the
;key just pressed was not on the numeric keypad, the statement
;will return a 0 (hex.) in the variable 'int%'.

        .FUNC   Keypad,0

        Open

        LDA     Kbdflag             ;Begin test to see if a special key was pressed
        CMP     #80
        BCS     Test                ;Is high bit set?. Yes a special key was pressed
        JMP     Done                ;                  No a regular key was pressed

Test    LDA     Kbd                 ;Get the keyboard byte
        STA     Kbdstb              ;Reset the keyboard strobe

        CMP     #0D                 ;Is it a 'RETURN' ?
        BEQ     Store               ;Yes - store it.
        CMP     #2D                 ;Is it a '-' ?
        BEQ     Store               ;Yes - store it.
        CMP     #2E                 ;Is it a '.' ?
        BEQ     Store               ;Yes - store it.

        CMP     #30                 ;Test to see if it is between
        BCC     Done                ;0 and 9
        CMP     #3A
        BCS     Done

Store   STA     Result              ;Save the answer

Done    Close                       ;Back we go!
        RTS

;                    - Function Bytetwo -
;
;Every time a key is pressed it generates two bytes of data. The
;first byte is easily found by reading the keyboard with standard language
```

Program listing continued on next page.

```
;statements. In BASIC the line ' 10 GET key$:key=ASC(key$) '
;will assign the value of the key pressed in the variable 'key'.
;In Pascal the lines of code    Read (Keyboard, ch)
;                               Char_val := ORD (ch)
;will assign the value of the key pressed in the variable 'Char_val'.
;Sometimes it becomes necessary to get the second byte of data, to test
;if a certain modifier key has been pressed. The BASIC statement
;'int% = EXFN%.Bytetwo' will return a value of the second byte of keyboard
; data as described on page 165 of the Standard Device Drivers Manual.
;In Pascal the lines of code    Function Bytetwo: INTEGER;
;                               External;
;will return the value of the second byte of keyboard data whenever the
;function is called.

        .FUNC   ByteTwo,0

        Open

        LDA     Kbdflag         ;Get the second byte of keyboard data
        STA     Result          ;Store it

        Close                   ;Come on back
        RTS

;                 -  Function Enter -
;
;This function will return a value of 1 if the 'ENTER' key on
;the numeric keypad has just been pressed. If the last key pressed
;was not the 'ENTER' key, the function will return a value of 0.
;The BASIC statement 'int% = EXFN%.Enter' will return the specified
;value in the variable 'int%'.

        .FUNC   Enter,0

        Open

        LDA     Kbdflag         ;Begin test to see if a special key was pressed
        CMP     #80
        BCS     Test            ;Is high bit set?. Yes a special key was pressed
        JMP     Done            ;                  No a regular key was pressed

Test    LDA     Kbd             ;If we get here a special key has been pressed
        STA     Kbdstb
        CMP     #0D             ;Was it the 'ENTER' key?
        BNE     Done            ;No - test is false

Store   LDA     #01
        STA     Result          ;Save the answer

Done    Close                   ;Back we go!
        RTS

        .END
```

Continued on next page

Key–Things: Program Listing #3

```
10    REM ****************************************************************
20    REM *                                                              *
30    REM *           Keypress Invokable Module Documentation            *
40    REM *                                                              *
50    REM *            (C) Copyright 1982, 1983 by ON THREE              *
60    REM *                                                              *
70    REM ****************************************************************
80    TEXT:title$=CHR$(15)+"----  KEYPRESS INVOKABLE MODULE  ----":GOSUB 300
90    PRINT:PRINT"     Before any Invokable Module can be used, it must be
   loaded into the":PRINT"system by the following Command Format:"
100    PRINT:PRINT")INVOKE KEYPRESS.INV":PRINT:PRINT"where KEYPRESS.INV can be
   the name of this or another Invokable Module.":PRINT:GOSUB 200
110    title$=CHR$(15)+"----  KEYPRESS INVOKABLE MODULE  ----":GOSUB 300
120    PRINT:PRINT TAB(8);"Select documentation on: ":PRINT
130    PRINT TAB(20);"1   Reading the keypad":PRINT TAB(20);"2   Reading the
   ENTER key"
140    PRINT TAB(20);"3   Two byte reads":PRINT TAB(20);"4   Quit":PRINT
150    PRINT TAB(8);"Which option ";:INPUT a$:x=ASC(a$)-48:IF x<0 THEN x=0
160    ON x GOTO 1000,2000,3000,180
170    PRINT TAB(8);"Please enter 1, 2, 3, or 4":VPOS= VPOS-2:GOTO 150
180    HOME:END
200    VPOS=24:PRINT USING"76c";"Press any key to Continue.";:GET a$:RETURN
300    PRINT CHR$(14):HOME:PRINT USING"76c";title$:PRINT:RETURN
400    PRINT:PRINT TAB(5)"The Command Format is:":PRINT
410    PRINT")key%=EXFN%.";funcname$:PRINT
420    PRINT TAB(5)"This command should normally be used right after a keyboard
    read such"
430    PRINT"as a GET or INPUT from BASIC or a READ or READLN from Pascal. It
    can also be"
440    PRINT"used from Pascal with the following statement.":PRINT
450    PRINT"FUNCTION ";funcname$;": INTEGER; EXTERNAL;":PRINT
460    PRINT"After being defined in your Pascal program it can be called just
   as any other"
470    PRINT"function. Remember it returns an INTEGER value so you can't
   assign it to a non-"
480    PRINT"integer type variable with out converting it first. From Pascal
   you must also"
490    PRINT"remember to L)ink before you can eXecute the file.":RETURN
1000    REM --- Keypad
1010    title$="-- Keypad --":GOSUB 300:REM Page 1
1020    PRINT TAB(5)"The KEYPAD Function returns the ASCII value of the last
   key pressed"
1030    PRINT"on the numeric keypad. If the last key pressed was not a key on
   the keypad"
1040    PRINT"the function will return a value of 0."
1050    funcname$="Keypad":GOSUB 400
1060    GOSUB 200:GOSUB 300:REM Page 2
1100    PRINT TAB(5)"You can also use this function from the Immediate
  Execution mode of BASIC"
1110    PRINT"by entering the following statements:":PRINT
1120    PRINT")key%=EXFN%.Keypad: PRINT key%":PRINT
1130    PRINT TAB(5)"I'll bet you entered the above line by pressing RETURN. If
  you did the"
1140    PRINT"screen will respond with:":PRINT:PRINT")0":PRINT
1150    PRINT TAB(5)"It returned a zero because the last key pressed (RETURN)
```

Program listing continued on next page.

```
   was not on the"
1160   PRINT"numeric keypad. Now try this! Enter the above line again but
   instead of pressing";
1170   PRINT"RETURN, hit the ENTER key on the numeric keypad. You should now
   be greeted with:"
1180   PRINT")13":PRINT
1190   PRINT"because the last key pressed (ENTER) was on the numeric keypad
   and its ASCII"
1200   PRINT"value is 13."
1210   GOSUB 200:GOTO 110:REM Go back to menu
2000   REM --- Enter
2010   title$="-- Enter --":GOSUB 300:REM Page 1
2020   PRINT TAB(5)"The ENTER Function returns a value of 1 if the last key
   pressed was the"
2030   PRINT"ENTER key on the numeric keypad. If the last key pressed was not
   the ENTER key"
2040   PRINT"the function will return a value of 0."
2050   funcname$="Enter":GOSUB 400
2060   GOSUB 200:GOTO 110:REM Go back to menu
3000   REM --- Bytetwo
3010   title$="-- Bytetwo --":GOSUB 300:REM Page 1
3020   PRINT TAB(5)"The BYTETWO Function returns the second byte of keyboard
   data of the last"
3030   PRINT"key pressed. The format of this byte is given in the Standard
   Device Drivers"
3040   PRINT"Manual, in Appendix G on page 165. Additional information can be
   found in the"
3050   PRINT"same manual on pages 135-137 and pages 47-49. Still a little more
   info. is on"
3060   PRINT"the next screen display."
3070   funcname$="Bytetwo":GOSUB 400
3080   GOSUB 200:GOSUB 300:REM Page 2
3100   PRINT TAB(5)"With a little programming expertise you can determine the
   state of all the"
3110   PRINT"modifier keys. For hints on how to do this, see the BASIC program
   KEYPRESS. It"
3120   PRINT"will show you how to read all the modifier key values.":PRINT
3130   PRINT TAB(5)"The value returned by the function Bytetwo will be a two
   byte integer in"
3140   PRINT"the range of 0-255. (Only the lower byte is used) The following
   table will give"
3150   PRINT"you an idea of how the various modifier keys are used.":PRINT
3160   PRINT"   Bit        7        6         5        4        3         2
       1       0"
3170   PRINT"**********---------+-------+---------+-------+-------+---------+
   --------+------+"
3180   PRINT"* Second * Special | Kybd | Closed | Open  | Alpha | Control |
   Shift | Any |"
3190   PRINT"*  Byte  *   key   |  on  |  Apple | Apple |  Lock |   key   |
   key  | key |"
3200   PRINT"**********---------+-------+---------+-------+-------+---------+
   --------+------+":PRINT
3210   PRINT TAB(5)"Bit 0 adds a 1 to the value returned if it is 'on', bit 1
   adds a 2 to"
3220   PRINT"the value if it is 'on', bit 2 adds a 4, bit 3 adds a 8..., and
   bit 7 adds"
```

Program listing continued on page 35

Basic - The Easy Way

by Earl Curlson

This is the first article in a series that is designed to teach you how to use Apple /// Business Basic. As such we need to get a few things straight. First of all, you don't have to learn how to program on the ///. Since many of you are business professionals you are probably using application packages that were designed to be easy to use. Thus, you may not need to learn Basic.

Then why do it? Well, one reason is that it might expand your horizons and make it a little easier for you to do your job. Secondly, you may be able to do something yourself that you would have had to pay someone else to do. Lastly, just for the fun of it! Yes, at times it can be fun! Watch for our 'Little Brick Out ///' in the April issue.

Before we get down to basics (pun intended!) you need to have a few things. For starters I'm going to assume that you have the Basic Language for the ///. We are going to use the two Reference Manuals quite extensively so you'll need those also. If you don't have either of these things, it's going to be awfully hard to keep up, so run on down to your favorite computer store and pick up a copy - it is worth it.

Now that everyone has a copy of one of the best little Basics in town, are we ready? Yes, you say - well almost! Before we start we should make a couple of copies of the Basic disk. You should NEVER (well, only to make the back - ups) use the copy that came with the package. Since the disk is not copy protected you're a fool if you don't make some copies.

How do I do that, you ask? Start by booting the Systems Utility Disk. When the menu comes up type 'D' and then 'F'. Now you're ready to format some disks. Insert a blank diskette that you haven't used before into the built in disk drive. STOP! Make sure there is nothing on the disk you are going to format, because it will erase anything you have on it! Enter '.D1' to format the disk in drive #1 and press 'RETURN'. Now enter the name you want the disk to be - '/BASIC' and press 'RETURN'. If everything goes well you will hear some grinding and whirring sounds coming from the drive. Don't worry that's normal.

Next, press the 'ESCAPE' key twice and then press 'F' and 'C'. Now we are ready to copy the files that let you use Basic on the ///. Insert the Basic disk into second disk drive and the disk that you just formatted into the internal drive. If you do not have a second floppy drive you will have to do some disk swapping. Now enter '.D2/SOS.=' and then press 'RETURN'. Here type '.D1/SOS.='. These commands will copy all the 'SOS' files from the Basic disk to the one you just formatted. Finally press 'RETURN' to begin the copy.

When it is finished you can use the 'Copy Volumes Command' to make copies of the disk you just made. Why not just use that command to make copies of the Basic disk? Simply because that disk has all sorts of files that we will not need yet, and it doesn't have much room on it.

> **'BASIC'..–<u>B</u>eginners <u>A</u>ll purpose <u>S</u>ymbolic <u>I</u>nstruction <u>C</u>ode**

Done at last, now let's get down to business and start learning how to program in Basic!

One last thing before we start. What the heck does 'BASIC' stand for? The next time somebody asks you, say - Beginners All purpose Symbolic Instruction Code!

Boot the disk that you just made, and in a few moments you will see a line of text at the top of the screen telling you Basic is 'up and running', and below the top line will be a right parenthesis ')' followed by a white block called the cursor. This is what is called the 'PROMPT' character. Whenever it appears, it means that the computer is waiting for you to type something.

The /// is very patient and will wait forever (or until you turn off the power) for you to type something. Try it - don't type anything for a couple of hours, or days- if you have the time. See, it waited all that time for you to enter something. If you are not too tired from that long wait, I suggest that we continue!

Open the Reference Manual (Volume #1 of course) and turn to page 4. Read the section on 'Entering Statements' and the

Continued on next page

one on 'Immediate and Deferred Execution' and then burn the manuals when you get disgusted. Wait, I was just kidding! The manuals are not really bad, it's just that they are what their name implies, 'Reference Manuals' and not much more. This column is the tutorial that will teach you how to use Basic, and down the road who knows? Maybe you will learn enough to read and understand the manuals.

After reading those two pages, you should understand that the statement 'PRINT' means something special to the computer. Remember, the computer is waiting for instructions from you, so when you entered 'PRINT "Hi there"', the computer did just what you told it to. The result was the words 'Hi there' on your screen.

As a test, type in 'SAY "Hello"' and then press 'RETURN'. Wait a minute! What is this '?SYNTAX ERROR' stuff? Well, the computer recognized the 'PRINT' statement above, but even though the Apple /// can produce sounds that mimic the human voice, the word 'SAY' is not in the vocabulary of this Basic.

We have reached an important point. There are certain words, commands if you will, that Basic understands and is able to use. 'PRINT' is one of them. Type 'HOME' and press the 'RETURN' key. Wow, what happened? We just saw that 'HOME' is another of the words that Basic understands. It erases the screen and moves the cursor to the upper left hand corner. Collectively, these statements that the computer understands are called 'Reserved Words'. Pages 236-237 (Volume #2) list all of the words that Basic understands.

. . . and then burn the manuals when you get disgusted.

If you read page 5 you will see something called 'immediate execution'. This is what happened when you typed in the 'PRINT' statements above. The computer did what you told it, it executed the statements immediately. Now enter the line on page 5, '10 PRINT "Hi again"' and press 'RETURN'. Type in 'RUN' and press 'RETURN' again. Just as the book says, what we just did is called 'deferred execution'.

If a statement that you type in has what they call a 'line number', it will not be executed until you type in the command 'RUN', which instructs the computer to do whatever you told it to do on the part of the line after the line number. You can thus build a very large number of statements that instruct the computer to do various things. This collection of lines is called a 'PROGRAM'.

These statements take up room in the computers memory, so if you have a program with a whole lot of statements (or a very small Apple!) the computer could give you the message '?OUT OF MEMORY', which means (appropriately), that there is no more room for the program.

Programs can be stored on a disk and later called up from the disk so that they may be executed. Type 'NEW' and then 'RETURN'. This statement erases the program that is in memory, in other words it wipes the slate clean, so you can enter a new program. I think you should now have the hang of pressing 'RETURN' when you enter a statement, so I will leave it out from now on.

Enter - '10 PRINT "Here I am."'. Now type in 'SAVE TEST'. The disk drive should make a few noises and then stop. You have just stored the program in memory onto the floppy disk. Why should you take my word for it? Check to see if it worked! How? just type 'CATALOG .D1'. This will produce a listing of all the files on the disk in the internal drive. You should see all the files that are stored on that disk, 'SOS. KERNEL, SOS.INTERP, SOS.DRIVER and TEST'.

Pages 128 to 135 will tell you all about how to manipulate files using Basic. Use the statement 'DELETE TEST' to remove the file 'TEST' from the disk in the internal drive. Now type - 'SAVE TERT'. Whoops! You meant to type 'SAVE TEST' but your finger slipped. If you really need to have the file named 'TEST' you can use the 'RENAME' command to change the files name. Type 'RENAME TERT, TEST'. Now if you catalog your disk, you will see that the name has been changed to 'TEST'!

CAUTION: Don't change the name of any of your files whose names start with 'SOS'. If you do, the next time you try to boot the disk, it won't! To protect against this type of disaster, you can use the command 'LOCK' to protect your files from being accidentally renamed or deleted. Type in 'LOCK TEST' and then 'DELETE TEST'. As you can see, the computer does not want you to try to delete the file 'TEST' because you have locked it, therefore it gives you the error message '?FILE LOCKED ERROR'.

If you catalog the Basic disk that came with the package (not the one you made), you will see a file with the name 'HELLO'. This name has a special meaning for Basic. When you boot a Basic disk, if

Continued on next page

there is a program with the name - 'HELLO' on the boot disk, the computer will load and then run it. Thus, you can develop a program that is automatically executed, whenever you start Basic.

Now do this for our program, type in 'UNLOCK TEST' (to allow us to change the name), then 'RENAME TEST, HELLO'. Now reboot the system by pressing CONTROL-RESET. In a few seconds you will see the line of text at the top of the screen, and then the disk will continue to stay on for a second while it is loading the 'HELLO' program. Finally, the line 'Here I am.' will appear, and Basic will be 'up and running'.

. . . you could get like me, bloodshot eyes and sore fingers !

A useful command is 'LIST', which as the name implies, lists all the statements in your program to the screen. The manual (pages 8-12) tells you how to format the way the listing occurs. If you find that a statement on one of the lines in your program is no good, you can remove it with the command 'DEL'. This deletes certain lines from your program. See the manual on pages 12-13 for an explanation of 'DEL'.

Before we go any further, we need to learn about editing the lines within your program. Say that you have typed in a number of statements and used the command 'LIST' to see them. In one of the lines you find that you have made a mistake. If you read pages 6-9, you will see that Basic allows you to change any mistakes in your program. You will also read that if you make a mistake in spelling one of the 'Reserved Words' the computer will give you an error message when you type 'RUN' and it tries to execute a 'PRITN' statement (or any other that you misspell).

So far we have only learned one of the 'real' commands of Basic - 'PRINT'. Now that we know how to do such things as 'LIST' a program, 'CATALOG' a disk, edit a program statement and perform some more of the utility functions, we are ready to do some real programming!

Type in the program on page 9 and type 'RUN'. It seems that the statement 'GOTO' causes the computer to start executing the statements at line #10. When it gets down to line #30, again the process is repeated. Forever. This is called a LOOP. Now look back and remember you typed 'RUN' just once and the computer has executed the program statements a couple of hundred times by the time you read this line.

Now you should see that a program can beat (rather easily) what you can do by hand, because it can repeatedly perform a bunch of lines at lightning fast speeds. Since you may be getting tired of seeing those lines over and over again, use the 'CONTROL C' command as described on page 9 to stop the program.

One thing that is neat and you should learn is the 'CONT' command. After you have stopped the run - away program using 'CONTROL C', you can use the command 'CONT' to have the computer pick up where it left off. Go ahead, try it. You should be greeted with a few hundred more lines of text.

I agree with the line on page 10 - you do deserve a break! When you come back, we will learn all about variables, how the /// knows the truth! and some decision making.

Boy, that was quick! You should not really spend so much time at the computer, you could get like me, bloodshot eyes and sore fingers! Well, back to the old grind. Remember why we use computers? That's right, because they do things better-faster -smarter (something like that) than humans.

You can use your Apple /// as a sort of high priced calculator by using the 'PRINT' statement with numbers. Type in 'PRINT 1+1'. Wow!-huh? You could do that one in your head. Now try 'PRINT 23 * 7'. Notice on the /// we use the symbol '*' for multiplication. Quite a few people use an 'X' to represent multiplication, but on the computer this could get confused with the letter "X" so we don't use it. Also, mathematicians like to use the dot ('.'), but this can get confused with the period, so on the Apple /// we use an asterisk.

Go ahead, try it. You should be greeted with a few hundred more lines of text.

So far we have dealt in what are called 'Whole Numbers'. These are numbers like '3, 87, -392 and 0'. This is the class of numbers that the computer understands that are called 'INTEGERS'. On the ///, these numbers can be in the range of -32768 to +32767.

All very nice, right? But what if you have to add numbers like '.3029, -1.9 and 93000'? The best way to answer that is to

Continued on next page

try it! Type in 'PRINT .3029 - 1.9'. Hey, it works! The /// can represent these numbers with fractional parts, with what are called 'Real Numbers'.

If you look back at the first thing that we typed into the computer, 'PRINT "Hi there"' you will see that we enclosed what we wanted printed inside of the double quote " marks. The Apple can also represent what are called 'Strings' by enclosing the text that you want printed in the double quote marks.

There is one more type of information that Basic can handle directly. That is the 'Long Integer' which we will discuss in depth at a later time. For now it's enough to know that Basic can use different types of information, and each type has its own name, - and certain rules that must be followed.

We have seen how the Apple can be used as a calculator, and since your /// costs a little bit more than a simple calculator it should, and does do all of the things a calculator can do. On some calculators you can save a number for later reference in what is called 'Memory'. Since your Apple has quite a bit of memory, there should be more than just one place to store numbers - and there is!

A long time ago, when I first learned how to use computers (on an Apple, of course!), I learned a name for these places to store numbers, and I'd like to pass it on to you now. We can put numbers in special places that we can call - 'pigeonholes'. To save the number 23, type 'LET M% = 23'. This is called an assignment statement, and it finds a place in memory to put the integer number 23 and assigns the name M% to its pigeonhole.

Now try the following line 'PRINT M%'. The computer will print the number 23. Note that we didn't use the double quote marks, because we want to print the number associated with the name 'M%', and not the string 'M%'. Try this now, type 'LET M% = 929' and then 'PRINT M%'. The /// should respond with a 929. What happened to the 23? It's gone! The pigeonhole can store only one value at a time, so whenever you change what's in it, the old value is erased.

Real numbers and strings can be saved in pigeonholes just like the integers shown above. Type 'LET N = 3.14159', and then 'PRINT N'. Likewise, you can store strings with statements like 'LET O$ = "This is a string"'. Experiment! Find out what you can and can't do - that's the only way your going to learn, so try different things. The worst thing that can happen, is your computer could give you an error message, telling that you made some type of error.

By now you are probably wondering why you had to type '%' after the 'M' and '$' after the 'O', but nothing after the 'N'. The answer is quite simple, whenever you want to store (or retrieve) some number or string, you have to specify what 'Type' of pigeonhole to look at. Try this, type 'LET Q% = 12345' and then 'PRINT Q'. You should get a zero for your troubles.

The number you saved was stored in the computers memory with the name 'Q', but just as importantly, with the type Integer. Whenever the '%' appears after the pigeonhole name, it tells the computer that the value is of the 'type' integer. Likewise, a '$' after the name indicates that it is a string. Nothing after the pigeonhole name indicates that it's a real number.

Type 'NEW' to erase any program that is currently in memory and type '10 LET A% = 0', '20 PRINT A%', '30 LET M% = M% + 1', '40 GOTO 20' and enter 'RUN'. The computer should respond with a screenful of numbers, each one being 1 bigger than the one before it. Looking at this program, we already know what lines 10, 20 and 40 do, so the only unsolved question is in line 30. This statement tells the computer to first take the value of what is in the integer pigeonhole whose name is 'A' and add 1 to it, and secondly store that number back in the same pigeonhole. When you get tired of the numbers growing bigger, you can press 'CONTROL C' to stop the program.

The Apple III has thousands that you can use.

A calculator may have just one pigeonhole, or place to store information. The Apple /// has thousands that you can use. The name the Basic manuals use for term 'pigeonholes' is 'Variables'. Don't get it confused with the term 'Variables' that is used in mathematics. They are not the same thing. Each one is just a place in the computers memory where one value - be it an integer, real or string - is stored.

You may want to read the manual from pages 38-49, there you will find a little more technical information about variables and the like. At this point I think you can learn a little short cut that will save you some typing in the future. The word 'LET' in any assignment statement is optional and may be omitted. Thus, the two statements 'LET M% = 23' and 'M% = 23' are

Continued on next page

exactly the same. Also, you can easily get tired of typing 'PRINT' line after line. The symbol '?' can be interchanged with the letters 'PRINT', to make your life a little easier. Try it - but remember to type out the 'PRINT' if you want the computer to print out the letters 'PRINT'. If you try to type in a 'PRINT "?", the computer will respond with a '?' instead of 'PRINT'.

The Apple /// knows the difference between what is true and what is false. To prove it, type in the following - 'PRINT 2 = 2'. The computer will respond with a '1', which indicates truth. Now try 'PRINT 2 = 9' and you will get a '0' because 2 does not equal 9. This is called 'Logical Operations' (They are, right?). Page 52 of the Basic manual lists all of the possible ways to create a logical operation. Practice using these, they will come in handy.

Say you wanted to print out the numbers from 1 to 10. One way to do this is called the brute force method. Enter '10 PRINT 1', '20 PRINT 2', etc. This may not seem so bad but if you wanted to print the first 1000 numbers, it would require 1000 statement lines. Our simple counting program above prints out numbers in an increasing order, but it does not stop at any particular number.

If you want to learn ...Basic, you will have to do all the examples that I give and more.

What we need is a statement that performs the GOTO 20 statement, if M% is, for example, less than 100 but doesn't perform the GOTO statement if M% is greater than or equal to 100. The answer to our prayer is the 'IF' statement. If the condition that is in the 'IF' statement is met, the computer will execute the instruction included in the 'IF' statement. If it is not met, the computer will skip the instruction that is after the 'IF' and goto the next line.

Enter the following program (Remember to type 'NEW' first!). '10 M% = 0', '20 PRINT M%', '30 LET M% = M% + 1', 40 IF M% < 100 THEN GOTO 20', '50 END'. This program will print out the numbers from 0 to 99 and then stop. It is an example of conditional branching, which is explained (somewhat) on pages 106-110 of the Basic manual.

The general form of the 'IF' statement is - 'IF expression THEN any statement'. First the expression is evaluated. If it is zero (false) the rest of the 'IF' statement is ignored, and the computer executes the next statement. If it is true (not = zero) the statement(s) after the 'THEN' are executed. This is a powerful feature which allows you to control the way the program operates.

Remember - If you want to learn how to program in Basic, you will have to do all the examples that I give and more. Therefore, you should read parts of the Basic manual now without help from me - only direction! Before we meet again, try to look over pages 2-53 and pages 102 through 110. Next month we will learn about 'FOR...NEXT' loops, functions, and subroutines, so you may want to look over the program below to sort of work ahead! ///

```
10     DEF FN Increment(num)=num+1
20     DEF FN Decrement(num)=num-1
30     DEF FN Neg.SQR(num)=SQR(num)*-1
40     DEF FN Rnd.100(seed)=INT(RND(seed)*100)
50     TEXT:HOME
60     VPOS=6:HPOS=28
70     PRINT"Negative square roots from 1 to 10"
80     VPOS=12
90     PRINT"Going down....Going up"
100     FOR loop.1=1 TO 10
110       value= FN Increment(value)
120       VPOS=loop.1:HPOS=1
130       PRINT FN Neg.SQR(value)
140       NEXT loop.1
200     FOR loop.2=10 TO 1 STEP-1
210       VPOS=loop.2:HPOS=15
220       PRINT FN Neg.SQR(value)
230       value= FN Decrement(value)
240       NEXT loop.2
250     VPOS=12:HPOS=24
260     PRINT"- They are the same both ways!"
270     FOR Column=1 TO 80
272       PRINT"-";
274       NEXT Column
276     VPOS=16:HPOS=9
280     PRINT"Some Random numbers"
290     VPOS=14
300     FOR loop.3=12 TO 77 STEP 12
310       GOSUB 60000
320       NEXT loop.3
330     VPOS=16:HPOS=48
350     PRINT"Some more Random numbers"
390     VPOS=14
400     FOR loop.4=12 TO 77 STEP 12
410       GOSUB 61000
420       HPOS=40
430       PRINT temp
440       NEXT loop.4
450     VPOS=21
460     PRINT"What's going on?"
470     PRINT"The second column isn't random!"
480     PRINT:END
60000     PRINT loop.3* FN Rnd.100(-loop.3)
60099     RETURN
61000     temp=loop.4* FN Rnd.100(-loop.4)
61099     RETURN
```

ON Visi...or /C maybe

by Louis Hanson

Welcome to a new column and (hopefully) some new insights into that marvelous tool - Visicalc ///.

I once read that many Visicalc users have only mastered about one third of Visicalc's features, so for the next few months (or until I change my mind!) I will concentrate on presenting useful functions that you may not know about.

As always, you should begin by saving anything you are currently working on. Next, boot the Visicalc program as described in the user's manual. To ensure your disks integrity, as soon as it finishes loading the program, carefully remove the boot disk from the internal drive and put it back in its protective jacket. Remember since you can't make a back up copy of the Visicalc disk, you must be especially careful with it.

For those of you hearty few who remember the early 13-sector Apple][version, you may recall - with a curse or two - one of the times you entered a long and complex formula and discovered (!) that you missed a parenthesis, or typed 'E4' when you meant 'D4'. Starting from scratch, you typed in the entire formula again, while trying to avoid new mistakes and remembering the original one!

Since as a group we are calm and conservative (right?) I'm sure you reacted the same way as I did when I got the new Apple /// version (1.1). I hope you didn't hit your head on the ceiling jumping for joy over all the neat new features.

Anyway, one of the new found features is the EDIT command. Never again will you have to re-type the entire value or label because of a simple mistake.

There are two ways to invoke the EDIT facility. First, say you are entering something and you see that you have made a mistake. On a clear worksheet type MISTEAK but don't press 'RETURN' yet. At this point you could press the 'ESCAPE' key a few times to delete the 'EAK' and then you could type in 'AKE' to correct it. Since we want to learn how to edit with more efficiency, just press 'CONTROL E'. The prompt line should now read '[Edit]: Label' and the edit line should display 'MISTEAK' with the cursor after the 'K'.

In the edit mode there are four ways to move the cursor. The ← and → move the cursor once to the left and right respectively. Pressing the ↑ moves it to the beginning of the entry while the ↓ moves it to the end.

To correct 'MISTEAK' press the ← twice. The cursor should now highlight (be over) the 'A'. Now press 'ESCAPE' once. This deletes the character to the left of the cursor ('E'). Like magic the 'E' disappears and the edit line will now hold 'MISTAK' with the cursor still on the 'A'. Now you can move to the end of the entry with one press of the ↓ . Type 'E' and then press 'RETURN'. Voila! it is fixed.

If you want to correct an error on a label or value that has already been entered into the worksheet, move the cursor to the cell you want to edit and type '/E'. Now correct the error(s) with the commands mentioned earlier and press 'RETURN' to enter the updated item.

If you attempt to enter an illegal expression (by pressing 'RETURN') in the edit mode, Visicalc will not accept it and will return to the edit mode with a beep and the cursor highlighting the offending character.

You can not change a cell from a label to a value or vice-versa in the edit mode. You can change a label to look like a value but because it is a label it will be worthless to your worksheet (and it may do some harm!).

Remember, anytime you enter something and it appears on the edit line you can invoke the edit mode by pressing 'CONTROL E'. If you are entering a file name or just looking through a directory you can edit what ever is on the edit line.

Now that I've done my monthly tutorial bit, lets look at something a little more interesting as we go...

Programming in VisiCalc

Most programmers are at least semiconscious of the fact that there are a few basic programming constructs, among them the IF... THEN... ELSE and the CASE statements found in most computer languages. Everyone may not know it but if you can create a template in Visicalc you are a programmer! You don't think so? Well to prove it I am going to show you how these programming constructs can be implemented in Visicalc ///. Don't get frightened away

Continued on next page

by the word 'programming', just sit back, read on and enjoy.

A simple CASE statement can be thought of as a construct that performs one of a number of tasks, according to an input value. In Basic this can be implemented by the 'ON xx GOSUB s1, s2, s3, ...,sn' statement. Here the value of 'xx' determines which one of the statement lines (s1,...) is executed next. In Pascal the statement 'CASE xx OF 1: Do_one; 2: Do_two; END;' does about the same thing.

Before we go on lets define a couple of terms. 'Function' and 'Procedure'. They sound simple enough but do you really know what they mean? Well, a Function is something that returns a value and a procedure is something that executes statements. Now these statements that are executed may change certain values, so a procedure can operate as a series of functions.

Getting back to reality, one of the often overlooked statements in Visicalc /// is the 'CHOOSE' function. You'll note that I said function and not procedure. Since Visicalc /// does not allow conditional execution of statements, these programming constructs can be only functions.

What the heck did I just mean? Don't worry, it's hard enough to understand let alone explain it. Imagine if you will a version of Visicalc in the future (near, I hope!). In a cell you may be able to type something like this. '@CHOOSE (c6,!F1,!F2, !F3)'. In cell F1 you may see '(!<A1"January Data", !DLOAD (A2..A31, ".D2/VISICALC/ JAN.DATA))'.

...one of the often overlooked statements in VisiCalc/// is the 'CHOOSE' function.

In my imaginary version (7.84B09ZZ9!), procedures are now a part of the language. The choose statement above now causes the statements at F1, F2 or F3 to be executed according to the data at C6. If C6 holds a 1 the procedure at F1 will be executed etc. At F1 the statement would first 'GOTO' cell A1 and store the label "January Data" in that cell. Next it would load the data from the file ".D2/VISICALC/JAN.DATA" into the cells A2 through A31.

Can you see what I mean? Todays Visicalc (sounds like a good name for a column - Ed.) can only return values from expressions while the next version of Visicalc (or its successor) may well allow the sheet to contain some kind of a rudimentary programming language akin to the Word Processing Language of Apple Writer ///.

...if you look back at our definition...you will see that you <u>can</u> program in VisiCalc!

Enough of procedures! and back to the choose function. The function 'CHOOSE (A1, 0,.2,.4,.6)' will return one of the values (0,..,.6) according to the value in the cell A1. If A1 holds the value 1, this function will return the value 0 - the first value in the list. If A1 holds a 2 it will return .2, etc.

Now enter the worksheet pictured on page **35** . At cell F3 enter '+D9*E9', at cell F4 enter '+D9/E9', at cell F5 enter '+D9+E9', and at cell F6 enter '+D9-E9'. Finally, at cell E11 enter the function '@CHOOSE(D8,F3,F4,F5,F6)'. If everything is correct you can now enter two numbers (any two will do). Put one at D9 and the other at E9. Finally make a choice and enter a number (1-4) at cell D8. As soon as you press the 'RETURN' key, the result of the calculation will appear at cell E11. We are done!

This template (program?) will perform one of four operations, dependent on the number or choice that the user types in. It performs these operations on the values the user gives, and thus if you look back at our definition of the CASE statement you will see that you can program in Visicalc!

That's all for this month, next time we will discuss some more of the less known (or understood) of Visicalc's features and maybe even take a look at the new Visicalc Advanced Version. Until then, if you have any questions or any comments about what you would like to see, write me in care of ON THREE. Remember - we are here to help you, so let us know what you want! ///

REVIEW ON: pfs: FILE & REPORT

by Bob Consorti

Organizing and managing information, that is what data base management is all about, and that is what the PFS series does. The question is how well and easy it works. This review will answer these questions and many more - hopefully enough to give the reader a good impression of what these programs can and cannot do. I say 'these' because to be a true Database management system, both are needed.

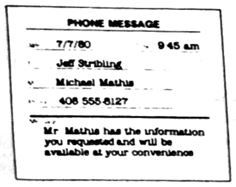
To start off we need to define a few terms. Data - Any and all information in the form of names, dates, places, events or even things like times. Database - Data that is stored in and by a computer. Data base management system (DBMS)- The software that allows a person to use and/or modify the data in the database.

FILE:

PFS: FILE is a free form DBMS whose versatility is only limited by the imagination of the user. I used the term 'free form' because the user defines the way the information is shown, and is not stuck in a pre-defined set of items. It operates on the principle that information is kept in forms. The user determines what the form looks like.

Below are some examples of forms that you can design:

Phone Messages


PHONE MESSAGE

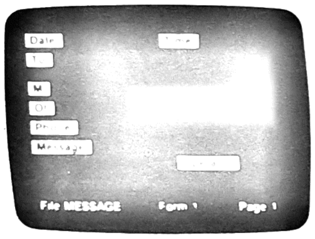
7/7/80 9 45 am

Jeff Stribling

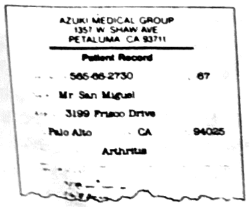
Michael Mathis

408 555-8127

Mr Mathis has the information you requested and will be available at your convenience



Patient Records


AZUKI MEDICAL GROUP
1357 W SHAW AVE
PETALUMA CA 93711

Patient Record

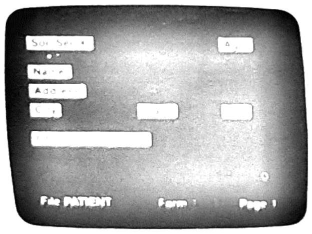
565-66-2730 67

Mr San Miguel

3199 Prisco Drive

Palo Alto CA 94025

Arthritis

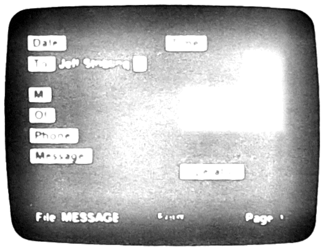


Once the user defines the form, it can be saved on disk and later be modified as needed. After the form is defined, the user can add new data to the database by filling in the form and storing it on disk. Once the information is stored in the database, it can be retrieved in a variety of different ways.

When you need to retrieve information, you use the blank form to indicate what you want to find. By filling in the form with retrieve specifications, the forms you wanted will be displayed.

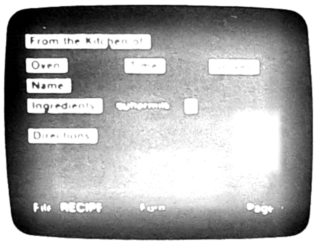
You can ask for all the items that exactly match a given set of characters (upper and lower case characters can be interchanged):

"find all the phone messages for Jeff Stribling"



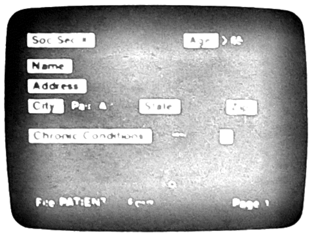
or all items that contain a certain set of characters:

"find all recipes that include buttermilk somewhere in the ingredients"



PFS: FILE also allows you to enter a retrieve specification for every item in the form. This feature gives you access to complex interrelationships between the items of information:

"find all patients over 65 who live in Palo Alto and suffer from arthritis"



The requested forms can be displayed on the screen and/or printed to a printer or disk file. The entire form or selected portions of it can be printed, formatted to your specifications.

As you may have guessed by now I do like PFS: FILE, and I do recommend it. One of its best features is its excellent user manual. Now before you run out to your

Continued on next page

favorite computer store and pick up a copy, let me say that it does have its drawbacks which I will discuss later on. Now I would like to ramble on for a bit about the PFS: FILE user's manual.

The manual is well written and has unusually good content. It's divided into a preface, a table of contents, an introduction, nine chapters, four appendices, a glossary, a very thorough index, and last but not least, a two page tip on the importance of the regular backing up of your files.

One feature of the manual that I like, is that in place of the smudgy pictures of the screen that some manuals give, this manual substitutes a framed, typeset image of the screen that is much easier to read. The one problem with this is, that while pictures don't lie, typesetters sometimes make mistakes. However, the quality of this manual is excellent and I have found only a couple of errors.

The manual was written to act as a guide and tutorial, and each of the nine chapters are structured along those lines. Each one has three parts, a description of the specific function defined in that chapter, an example of how to use it, and a summary. I have found that this works very well, when you need to look up something the chapter summary can be a big help.

The first three chapters cover the storing of information: telling how to design a file, add information to the file and to copy information from one file to another, respectively. The next three chapters are on the retrieval capabilities of the program. You can search or update, print, and remove forms from a file. The last three chapters are also useful, they tell how to send special characters to your printer, cataloging your disks and changing the design of your forms.

The appendices are also worth mentioning. They A: Give all the error messages that it is possible to get using the system along with a detailed description and list of possible corrective actions you can take to recover from the error. B: Give users a summary of the special control keys that are used in the program. C: Tells how to determine how many forms you can store on a disk. D: Defines the relationship to Apple /// system software, including information on how to rename and delete files, format data disks and using the System Configuration Program to alter the drivers on the PFS: FILE boot disk.

As you can see, the manual is quite good. The program isn't that bad either. For the next few line I will tell about the programs capabilities, and drawbacks. Yes sports fans it won't wash the dishes or walk the dog, but it does a good job at delivering what it promised - to organize and manage your information.

Remember how PFS: FILE handles information? Well, what happens if you want to design a form that is bigger than the screen? The answer is, that each screen of information in the form is just one page in a sort of electronic booklet. The user can move back and forth between the pages very easily by using the control keys 'CTRL N' to bring up the next page, and 'CTRL P' to bring up the previous page.

At the bottom right hand corner of the screen is the current page number. If there are more pages to the file, it will have an asterisk next to it. I have found a slight error to this that may give you some grief. On a multiple page form, on pages other than the first, the asterisk will at times be there and at other times be absent. I have found no way around this problem and must assume it is a programming problem.

Since each form can be up to 32 pages long and there can be up to 100 items per page, you can develop quite complicated forms. PFS: FILE uses the features of the /// well. The cursor control keys are fully operational, and it uses the 'ENTER' key to signal to continue with the selected function. Another nice feature for those of you who have used PFS on the Apple][is you can also use 'CTRL C' in place of the 'ENTER' key.

The manual was written to act as a guide and tutorial...

Ever wonder why those nuts at Apple put a 'TAB' key on the keyboard? Well, after you work with PFS for a while you will see how useful it is. This program uses the 'TAB' key to quickly position the cursor between items. Holding the shift key down and pressing 'TAB' moves the cursor backwards. Try it, I think you'll like it!

If you ever decide that the information you just put on a page is completely wrong and you want to erase it, do just that! Press 'CTRL E' and the page currently displayed will be erased. Additional control functions that can be accessed at any time are - 'CTRL D' which enters the current date at the position of the cursor,

Continued on next page

and 'CTRL T' which enters the time. If your system doesn't have a clock installed these last two 'control' functions will return meaningless data. Press 'CTRL O' any time a form is displayed on the screen and the print options menu will come up. This allows you to print out selected forms during a search or update operation.

When doing a search or when adding new information pressing 'CTRL N' usually brings up the next page of the form. But what happens when you need to add some extra information that you didn't leave room for on the form? Press 'CTRL N' when at the last page of the form, and a new screen will appear, appropriately labeled 'ATTACHMENT:'. You can continue to add attachment pages using 'CTRL N' until you run out of room on the diskette.

...it allows access to any given form within 3—5 seconds.

Once you have entered information into a form and stored the file, you can search through the file to find forms that interest you. While searching, you can update forms by making changes in the information. This is one of the strong (and weak - as explained later) points of PFS: FILE. To use the search / update feature you must enter items called retrieve specifications.

These are divided into four categories 1: Full item match - exactly as it sounds, it matches forms with the exact specifications you enter. A nice point is that it doesn't matter if you type the spec. in upper or lower case. 2: Partial item match - it matches forms that have an occurrence of your specification in them. This is helpful when you can't remember the exact phrasing of an item you entered. 3: Numeric item match - again just as it sounds, it will match forms with the spec. that you give. You can search for numbers >, <, or = to a given value, or it can search for numbers that are in a particular range like =1..9, which looks for numbers that are in between 1 and 9 inclusive. 4: The NOT feature - any of the first three specs. can be preceded by a slash '/'. This reverses the meaning of the retrieve specification.

When searching, PFS: FILE treats the first item of the form slightly differently than the others. Using full item match on the first item, the program is able to go directly to the desired form on the disk file. For other items in the form, it must search through each form in the file, which takes longer. This is good because it allows access to any given form within 3-5 seconds.

The problem is if, say you have a mailing list with 1000 SMITH's on it and you want to find the one living on 123 Any Street. If you set up the file so that SMITH is the first item, the program will have to do a sequential search through all of the forms until it finds the right one. This could take some time for a large file. Knowing my way around database systems I find this unacceptable because there are many hashing methods that allow inclusion of a secondary key, so you can go directly to the Smith living on 123 Any Street. Of the few things I dislike about this system that is the greatest. I do hope that the program authors, John Page and D.D. Roberts try to fix this little problem.

Another one of PFS: FILE's problems is that it does not allow the database to spread over more than one disk. While hard disk owners can have up to 32000 forms in a file, users of the standard floppy disk can only store up to 1000 forms per diskette. A very welcome enhancement would be the ability to have multiple disks per file.

One way around this is to use the COPY function of the main menu to copy the form design from a full disk to an empty one, thus giving you a new diskette to store information on. You can also use this to copy selected forms from one file to another file using the retrieve specifications mentioned above. Another option of this function is to convert PFS files created on an Apple][to the Apple ///. Because the internal file structure of the]['s files are different from those on the ///, the old files are not usable on the Apple /// without converting them first.

This ability to copy data from an Apple][file is good, but the program can not copy information from selected parts of a text file, or from a Visicalc data file. The programs usefulness, and its potential audience would be increased if these capabilities were added.

The PRINT function allows you to print selected forms and / or portions of forms according to the format specifications that you specify. It also lets you print the forms in a sorted order, like by zip code or state. Once you have defined a print specification, it can be stored with the file, and used again in the future without having to re-enter those specifications. Again PFS: FILE uses the advanced features of the Apple /// in allowing the output to go to almost any device that is configured

Continued on next page

into your system. Thus you can print to a disk file if you wish.

I said almost any device because it won't print to the console, which is a bad design feature. There are times when you may not want to waste the paper, or the time in printing to a disk file and then booting up a word processor to look at the resulting output. If you could print to the console, time and money would be saved, and after all that's why we have computers - right? It's just another minor problem that I hope is corrected in a new release.

The format specifications for the print option are quite simple and easy to learn. 'X' prints this item, and then advances to the next line, '+' prints this item but doesn't advance to the next line after printing it, instead it skips two spaces. This allow you to print more than one item per line. You may also sort the printout based on the value of a certain item in the form.

While very easy to learn, they are at times too simple. You can only sort one way - in ascending order, and only on one key, thus there are no multiple key sorts. You can do mailing lables but the print format is somewhat restricted, you can't add commas, or other special characters using the print specs., only spaces. Again, this is a minor irritant that could be easily corrected and I hope will be, soon.

At times you may decide that a particular form in your database is outdated and needs to be removed. PFS: FILE allows you to remove forms that you are no longer interested in. There are two ways to remove a form, 1: during a search / update if you find a form you want removed, simply press 'CTRL R' and the program will ask you if you're sure you want to remove it. 2: use the REMOVE option of the main menu and you can selectively remove a number of forms at once using the appropriate retrieve specs.

The REMOVE function has one annoying feature that is not as easily corrected. When you remove a form, its space is reclaimed for future use, but its number goes into never never land and can't be used again. Thus over a period of months and many additions and removals, your file may have only a few thousand forms, but because the form number is incremented each time you add a new form and is not decremented when you remove forms, the form number may reach 32000 which is the maximum. The program doesn't know that space is available and will not store new forms in the file.

This is corrected by only one method. Use the CHANGE design option without making changes to the form, and the forms will be renumbered as they are copied, thus freeing form numbers for re-use. The fault with this is that in a few thousand form file, it will take hours to complete.

A very handy feature is the ability to send special characters to the printer before each print run. Some printers require that a special character sequence be sent before printing starts. For example sending a 'CTRL O' to an Epson MX-80 or MX-100, causes it to print in the compressed character mode. The program 'echoes' the characters you enter in English form so you can verify that you have entered the correct characters. Another minor bug here, you can't save the sequences you create, and must re-enter them whenever you reboot the program.

...in a few thousand form file, it will take hours...

Another nice feature is the ability to list the files stored on any of your disks. Just enter the directory name (.D1, .D2, .PROFILE, etc.) and the files will be listed along with their type (Datafile, Binary ...). Again we have a minor bug that is easily correctable. There are times when I want to know how big a file is, and how many blocks are left any on the disk etc. It's annoying to have to boot a BASIC or PASCAL disk just to get a full directory listing.

Next month we will publish a Pascal program that lists the COMPLETE directory information. We encourage all the software developers who program in Pascal to use it, it will make your programs much more user friendly. Along those lines we will also publish the complete listing of the directory structure of Apple /// SOS disks.

Perhaps the most useful function of PFS: FILE is the ability to change the design of a form even if it already contains data. This allows you to include new items or delete old ones in the design of the form without having to manually re-enter all the data again. However you can not change the name of an item. If it was 'NAME:' on the old design and 'NAMES:' on the new design, the data from that item will not be copied.

Items can be moved to different places on the form, even different pages with the limitation that you can't change more than four pages of a multiple page form at once. While quite sophisticated, it has room for improvemment and hopefully in a future re-

Continued on next page

lease you will be able to map from one item to another.

As I said before, the manuals are very good with only a few typos. Below is a listing which is complete to the best of my knowledge. If anyone else knows of any more, please write ON THREE and Software Publishing Corporation. We will print it for all to see, and they will fix it.

Page 4-13: 1st screen '> $1850' should be '<$1850'.
Page 5-13: They are not the right labels and they're not sorted by zip.
Page D-2: The information about formatting the PROFILE hard disk could give the reader the impression that you should format the disk to get a new volume name. You can use the Rename a volume option to accomplish this. Naive users may re-format their hard disk and lose all the information on it.
Page D-3: The note on renaming a Corvus hard disk, '.Profile' is incomplete, see later in the review of PFS: REPORT for an explanation.

Before I finish up I would like to mention a few things that haven't been previously covered. PFS: FILE doesn't claim to be a DBMS, and that's good because it isn't really one. What it is, is just what it says - Personal Filing System. A real DBMS should include things like data security and input checking. But I think it has to be said that in designing a program that is as easy to use as PFS: FILE is, I think certain trade-offs had to be made. However, with the few exceptions mentioned in this article, the resulting program is very good, and well suited to managing your information needs.

Below are some timings I made of the programs performance. Beware, they will change depending on the number of active drivers, and the size and speed of any disk drives you may have in your system. I used a standard Apple /// floppy in my tests. Hard disk drives (of course) will give much quicker operations.

Boot Time: 45 seconds with given drivers.
Copy Design: 33 seconds for a 1 page form.
Change Design: 1:15 for a blank 1 pg. form.
Copy 100 forms (Design below): 17 minutes.

```
Name:
Address:
City:                State:   Zip:
```

Print 100 labels to disk: 1:50.
Disk Capacity (Above design):
Standard Disk: Approx. 925.
Modified Disk: Approx. 990.
(See Article 'Disk Pak1' for info.)

REPORT:

A good DBMS must have some kind of a report generator, and the PFS series of information management has PFS: REPORT. It can be used to summarize the data that PFS: FILE stores. It will produce tabular reports consisting of up to 20 vertical columns, each of which corresponds to an item from a PFS: FILE file. It will sort the rows alphabetically or numerically, and you can perform calculations on various file information to produce averages, sub-counts, totals and more.

This manual is also well written. It is divided into a preface, a table of contents, an introduction, three chapters, four appendices, a glossary and an index. Like the PFS: FILE manual, this one was written to be both a guide and a tutorial.

The first chapter tells you how to print a report, starting with a very simple form and slowing increasing the complexity until you're adept at printing reports with multiple column calculations and even derived columns.

Once you create a report you can use the information in chapter 2 to pre-define a report. Thus, if you have to give a report at regular intervals you only have to design it once. These pre-defined report specifications can then be stored on disk and called up when needed.

...you can send the output to a printer, disk file or the console.

A handy feature is the 'Set new headings' function. Since the column widths on the report are determined by the number of characters in the widest heading, you may sometimes get a report like this:

```
Grade Point Average
-------------------
3.2
4.0
2.6
```

Using this function you can change the column heading to something more reasonable like, 'GPA'.

When printing a report you can specify any or all of the forms, using the same retrieve specification format as PFS: FILE,

Continued on next page

and you can send the output to a printer, disk file or the console. You can set up a sequence of special characters to be sent to your printer before each print run, but there is no menu option for this as in PFS: FILE. You must press 'CTRL S' at the main menu level.

This is a bad design feature that should be corrected in a future version. No program should force the user to remember control sequences like this one without having some type of on-line 'HELP' system.

...columns are automatically formatted so that decimal points and commas are aligned.

The information printed in each column can be treated differently. For example, one column of numbers can be averaged, with an average printed out at the end. Another column of numbers could be totaled up, or subtotaled whenever the item in column 1 changes. All numeric columns are automatically formatted so that decimal points and commas are aligned. Subaverages, counts, subcounts and page breaks can all be specified when defining the way the report is to be printed.

The best part of this program is its ability to create derived columns in the report. Derived columns are calculated from information in other columns. This means that you can total up information from other columns, perform complex calculations and print the result in a new column that you can name.

All in all, it is a good program that you can use to create presentation quality reports from the information in your PFS: FILE files. However, its sorting capabilities could be improved greatly. By being able to sort on only the first and second field alphabetically if it contains alpha. information, and in descending order if it is numeric, the program boxes the user into a pre-defined set of rules, and that should be corrected.

You may not believe this but here goes: If you want to do sorting on a moderate sized file (> 200 blocks), you not only have to have a second drive - but the disk in the drive MUST be named 'SORTWORK'!! That bit of information was found buried in one of the appendices. If you have a PROFILE hard disk, you will not encounter this problem because the program will create the temporary work file on the PROFILE. But, even if you have a second disk drive with a blank disk in it - the program will not be able to do the report until you insert a disk with the name 'SORTWORK'.

This is why the manual says that if you have a Corvus, you must rename it - '.PROFILE'. If you don't, the program will not be able to find a work space - even though a 20 megabyte drive may be connected and completely empty. Talk about a bad program design!

PFS: FILE is based on the 'free form' system, whereby the user chooses how his or her information is placed, and I see no reason why PFS: REPORT shouldn't be the same way. Also, by having NO capability of listing the files on a diskette the program is unnecessarily limited and forces the user to remember the exact file name. Since Software Publishing Corporation has very good customer support I hope they listen to the ideas mentioned in this review for improving their software.

Below is a list of timings I made of the program. You can use PFS: REPORT with a single disk drive, but you can't sort the report or do any of the functions that are dependent on the sort if the file is bigger than approx. 200 blocks.

Boot Time: 42 seconds with given drivers.
Print a report of 100 forms (Labels) to floppy disk: 3:30
(Sorted by state & zip code & sub-counted each time the state field changed)

Software Publishing Corp. has a very good program support policy...

As a final note: when you mail in your registration card you will be sent a copy of the "PFS: File Forms Sampler" disk. This contains many pre-made forms that will speed up your mastering of the program. However, it is not ready yet - but a quick call to the company revealed that when it is ready, all owners who sent in their registration card will get one. The timetable that was mentioned was mid to late Ferbruary. Software Publishing Corporation has a very good program support policy, if you call the main office, they will answer most any question that a novice or expert can come up with.

Wrap-up on next page

Equipment used in the review:

> 128K Apple ///
> 1 external floppy drive
> MX-100 printer, out of
> the serial port.

Program: PFS: FILE for the Apple ///.
Version: Program - B.01, Manual - Rev. C
Contents: Program Diskette, Data Storage Diskette, User's Manual.
Cost: $175.00

Program: PFS: REPORT for the Apple ///.
Version: Program - B.00, Manual - Rev. B
Contents: Program Diskette, SORTWORK Diskette, User's Manual.
Cost: $125.00

For both programs

Programming language: Pascal & assembly
Operating System: Standard SOS
Copy Protected: Yes
Disk Warranty: 90 days
Time to send a replacement disk that I mailed using an overnight mail service:

6 Business days

Backup disk included: No
Cost of backup disk: $15.00
Time to send a backup disk that I sent using regular mail:

10 Business days

The Bottom Line

PFS: FILE

Performance: Good
Documentation: Excellent
Ease of Use: Excellent
Error Handling: Good - Excellent
Over All Rating: A -

PFS: REPORT

Performance: Fair - Good
Documentation: Good
Ease of Use: Good
Error Handling: Good
Over All Rating: B -

Continued from page 28

```
      A          B          C          D          E          F          G          H

 1 What type of operation is desired ?
 2
 3           1 - for multiplication
 4           2 - for division
 5           3 - for addition
 6           4 - for subtraction
 7
 8 Enter your choice ( 1 - 4 )
 9 Enter the two numbers -->
10 ------------------------------------------------------------------------------
11 The result operation number  is ----
```

Continued from page 21

```
3230    PRINT"128 to the value returned if it is 'on'.":PRINT
3240    GOSUB 200:GOSUB 300:REM Page 3
3300    PRINT TAB(4);"NOTE: Contrary to what the manual shows on page 135 that
  the RETURN key is"
3310    PRINT"a 'Special' key, this function will not return a value with the
  'Spcl Key' bit"
3320    PRINT"set if RETURN is pressed. I believe that they meant that the
  RETURN key is a"
3330    PRINT"special key (as you well know!)."
3340    GOSUB 200:GOTO 110:REM Go back to menu
```

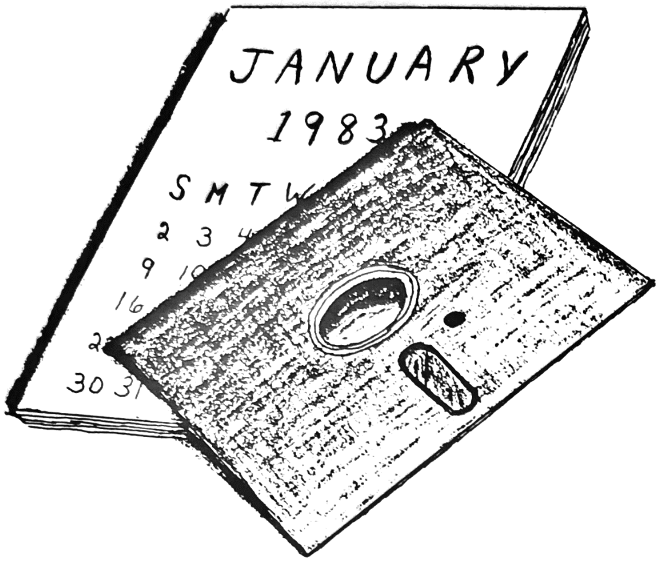
Three Shorts - Fini!

If you have gotten this far I think you deserve a rest. But before you go and lay down, type in the following programs. They will dazzle your eyes and soothe that headache you're getting! ///

```
0    REM ***************************************************
1    REM *  Brian's THREEme                                *
2    REM *  ---------------                                *
3    REM *  This is the Apple /// version of the Apple ][  *
4    REM *  program "Brian's Theme".  It will produce a neat *
5    REM *  "moire" type pattern on your Apple ///'s high  *
6    REM *  resolution graphics screen.  To use it, type in *
7    REM *  the program, and make sure that "BGRAF.INV" is  *
8    REM *  available.  Type "RUN", sit back and enjoy!    *
9    REM ***************************************************
10    ON ERR INVOKE"/BASIC/BGRAF.INV"
20    PERFORM initgrafix:OFF ERR
100    xhigh%=559:yhigh%=191
110    mode%=2:buf%=1
120    PERFORM grafixmode(%mode%,%buf%)
130    PERFORM grafixon:PERFORM fillport
140    fill%=0:pen%=15
150    PERFORM pencolor(%pen%)
199    ON KBD GOTO 1000
200    a%=RND(1)*xhigh%:b%=RND(1)*yhigh%
210    i%=RND(1)*3+1
220    FOR x%=0 TO xhigh% STEP i%
230      FOR s%=0 TO 1
240        PERFORM pencolor(%s%*pen%)
250        PERFORM moveto(%x%+s%,%fill%)
260        PERFORM lineto(%a%,%b%)
270        PERFORM lineto(%xhigh%-x%-s%,%yhigh%)
280        NEXT s%,x%
290    FOR y%=0 TO yhigh% STEP i%
300      FOR s%=0 TO 1
310        PERFORM pencolor(%s%*15)
320        PERFORM moveto(%560,%y%+s%)
330        PERFORM lineto(%a%,%b%)
340        PERFORM lineto(%0,%192-y%-s%)
350        NEXT s%,y%
399    TEXT:END
1000    IF KBD=27 THEN POP:TEXT:END
1010    ON KBD GOTO 1000
1020    RETURN
```

```
0    REM ***************************************************
1    REM *  Brian's Colors                                 *
2    REM *  --------------                                 *
3    REM *  This is a color version of "Brian's THREEme".  *
4    REM *  Since the Apple /// has good graphics potential, *
5    REM *  this program uses those magnificent colors to  *
6    REM *  show the nice "Moire" type patterns.  To use   *
7    REM *  the program, make sure that "BGRAF.INV" is avail- *
8    REM *  able on one of your diskettes.                 *
9    REM ***************************************************
10    ON ERR INVOKE"/BASIC/BGRAF.INV"
20    PERFORM initgrafix:OFF ERR
100    xhigh%=139:yhigh%=191
110    backround%=0:mode%=3:buf%=1
120    PERFORM grafixmode(%mode%,%buf%)
130    PERFORM grafixon:PERFORM fillport
140    fill%=RND(1)*15:pen%=RND(1)*15
150    IF pen%=fill% THEN 140
160    s%(0)=fill%:s%(1)=pen%
199    ON KBD GOTO 1000
200    a%=RND(1)*xhigh%:b%=RND(1)*yhigh%
210    i%=RND(1)*2+1
220    FOR x%=0 TO xhigh% STEP i%
230      FOR s%=0 TO 1
240        PERFORM pencolor(%s%(s%))
250        PERFORM moveto(%x%+s%,%0)
260        PERFORM lineto(%a%,%b%)
270        PERFORM lineto(%xhigh%-x%-s%,%yhigh%)
280        NEXT s%,x%
290    FOR y%=0 TO yhigh% STEP i%
300      FOR s%=0 TO 1
310        PERFORM pencolor(%s%(s%))
320        PERFORM moveto(%xhigh%,%y%+s%)
330        PERFORM lineto(%a%,%b%)
340        PERFORM lineto(%0,%yhigh%-y%-s%)
350        NEXT s%,y%
399    GOTO 100
1000    IF KBD=27 THEN POP:TEXT:END
1010    ON KBD GOTO 1000
1020    RETURN
```

```
0    REM ***************************************************
1    REM *  Patterns - Part 1                              *
2    REM *  -----------------                              *
3    REM *                                                 *
4    REM *  This program will draw a colorful pattern on your *
5    REM *  Apple ///'s graphic screen.  To use, type in the *
6    REM *  program and make sure that "BGRAF.INV" is on one *
7    REM *  of your disks.  See next months issue for Part 2. *
8    REM *                                                 *
9    REM ***************************************************
10    ON ERR INVOKE"/BASIC/BGRAF.INV"
20    PERFORM initgrafix:OFF ERR
100    xhigh%=139:yhigh%=191
110    backround%=0:mode%=3:buf%=1
120    PERFORM grafixmode(%mode%,%buf%)
130    PERFORM grafixon
140    PERFORM fillcolor(%backround%)
150    PERFORM fillport
199    ON KBD GOTO 1000
200    FOR pencolor%=1 TO 15
210      PERFORM pencolor(%pencolor%)
220      x%=INT(RND(1)*10):IF x%=oldx% THEN 220:ELSE oldx%=x%
230      FOR a%=0 TO xhigh%/2 STEP x%+3
240        PERFORM moveto(%a%,%0):PERFORM lineto(%xhigh%/2-
a%,%yhigh%/2)
250        PERFORM moveto(%xhigh%-a%,%0):PERFORM
lineto(%xhigh%/2+a%,%yhigh%/2)
260        PERFORM moveto(%a%,%yhigh%):PERFORM
lineto(%xhigh%/2-a%,%yhigh%/2+1)
270        PERFORM moveto(%xhigh%-a%,%yhigh%):PERFORM
lineto(%xhigh%/2+a%,%yhigh%/2+1)
280        NEXT a%
300      FOR b%=0 TO yhigh%/2 STEP x%+2
310        PERFORM moveto(%0,%yhigh%/2-b%):PERFORM
lineto(%xhigh%/2,%b%)
320        PERFORM moveto(%xhigh%/2,%b%):PERFORM
lineto(%xhigh%,%yhigh%/2-b%)
330        PERFORM moveto(%0,%yhigh%/2+1+b%):PERFORM
lineto(%xhigh%/2,%yhigh%-b%)
340        PERFORM moveto(%xhigh%,%yhigh%/2+1+b%):PERFORM
lineto(%xhigh%/2,%yhigh%-b%)
350        NEXT b%
360      NEXT pencolor%
370    PERFORM xfroption(%6):PERFORM fillport:PERFORM fillport:
PERFORM fillport:PERFORM fillport
380    GOTO 200
1000    IF KBD=27 THEN POP:TEXT:END
1010    ON KBD GOTO 1000
1020    RETURN
```



Disk Of the Month

Calling all you busy professionals, would you like to have the programs in this months issue? What's that? You don't have the time to type them in yourself? Well, just buy the disk!

This disk includes all the programs included in this issue (Extra Disk Space, Key-things, Graphic-Demos) and more!

To discourage piracy, we have priced these disks so low, that everyone can afford one.

Buy one now for the introductory price of **$9.95** *(Plus $1.50 for postage and handling).*

Group rates are as follows:

2 - 9 disks:	**$7.50** *apiece + $3 total shipping*
10 - 24 disks:	**$7.00** *apiece + $4 total shipping*
over 24 disks:	**$6.50** *apiece + $5 total shipping*

Group rates must have one mailing address. Please use the attached envelope for orders.

ON THREE O'Clock

How would you like a working clock/calendar for your Apple III? Just as it was originally intended, a plug in clock chip with a battery backup.

Extremely easy to install and adjust, this is the one you have been waiting for!

This package contains comprehensive instructions and a Six Month Warranty ! Try to get that deal anywhere else!

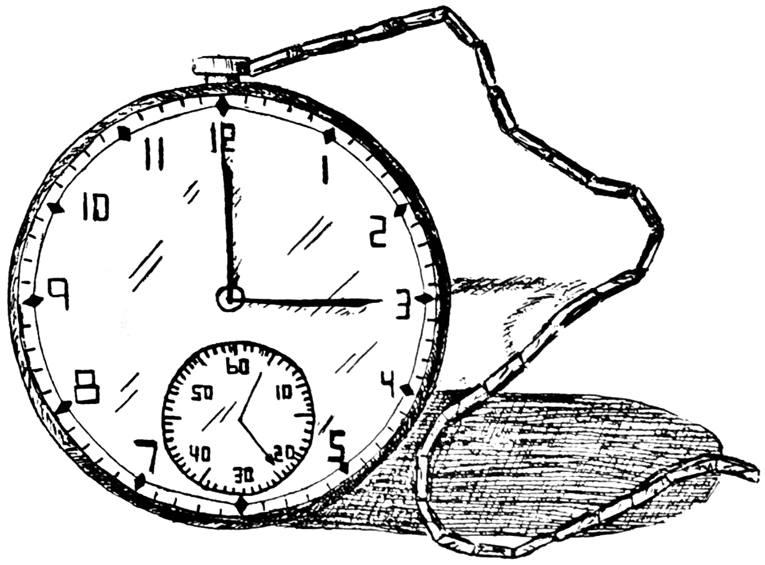
What's the best part? - The price! While others are selling theirs for **$60** *and up, we have broken the* **$50** *barrier. Heck, we broke the* **$40** *barrier!*

For only **$39.95** *(plus $2.50 for postage and handling) you can get the best little clock in town!*

Group rates are as follows:

2 - 9 clock sets:	**$36.50** *apiece + $5 total shipping*
10 - 24 clock sets:	**$33.25** *apiece + $7 total shipping*
over 24 clock sets:	**$31.00** *apiece + $9 total shipping*

Group rates must have one mailing address. Please use the attached envelope for orders.



BULK RATE
U.S. POSTAGE PAID
VENTURA, CA.
PERMIT NO. 90